DeepSee MDX Reference

Version 2018.1
2019-09-20

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

DeepSee MDX Reference
Caché Version 2018.1 2019-09-20

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

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 the basic elements used in DeepSee queries: cubes and subject areas. It also describes how to define listing groups.
- *Advanced DeepSee Modeling Guide* describes how to use the more advanced and less common DeepSee modeling features: computed dimensions, unstructured data in cubes, compound cubes, cube relationships, term lists, quality measures, KPIs, plugins, and other special options.
- *Using MDX with DeepSee* introduces MDX and describes how to write MDX queries manually for use with DeepSee cubes.
- *Tools for Creating DeepSee Web Clients* provides information on the DeepSee JavaScript and REST APIs, which you can use to create web clients for your DeepSee applications.

The following books are for both developers and users:

- *DeepSee End User Guide* describes how to use the DeepSee User Portal and dashboards.
- *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.

Also see the article *Using PMML Models in Caché*.

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*:

  ```
  name
  ```

  You can use this form if *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 InterSystems 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*:

  ```
  [name]
  ```

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

- (Only for members) A *member key*:

  ```
  &[key value]
  ```

  For information on key values, see "Key Values."

Note: *name* is not case-sensitive, but *key value* is.

Important: Note that *name* and *key value* are not expressions. That is, you cannot replace them with string-valued expressions.

Comments

This section discusses comments in DeepSee MDX queries and in stand-alone MDX expressions used within model definitions.

Comments

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

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

You can include comments of the following forms as separate lines within an MDX query or expression.

```
-- comment
```

Or:

```
// comment
```

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 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: `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: `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.

- A member expression, such as `[gend].[h1].[gender].[female]`

  Note that the value of a member expression depends upon the measure that is currently in use. By default, this expression evaluates to the number of records that belong this member. In contrast, if a specific measure is in use, this expression evaluates to the aggregate value of that measure across those records.

- The MDX identifier for a dimension, such as `[gend]`

  Note that the value of this expression depends upon the measure that is currently in use. By default, this expression evaluates to the number of records in the cube. In contrast, if a specific measure is in use, this expression evaluates to the aggregate value of that measure across all records in the cube.

- A reference to a pivot variable that contains a numeric value. To refer to a pivot variable, use the following syntax:

  ```
  $VARIABLE.variablename
  ```

  Where *variablename* is the logical variable name. Do not enclose this expression with square brackets. This syntax is not case-sensitive; nor is the pivot variable name.

  For information on defining pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

Uses

You can use numeric expressions in the following ways:

- As a numeric argument to many MDX functions. For example:

  ```
  AVG(diagd.MEMBERS, MEASURES.[%COUNT])
  ```

- As an element of a set.
- As the definition of a calculated member (in this case, a measure).

Examples

This section shows examples of some of the less common kinds of numeric expressions. The first example shows that a member expression has a numeric value:

```
SELECT gend.h1.gender.female ON 0 FROM patients
```

```
                                    Female
                                        526
```

The next example is a variation of the preceding:

```
SELECT gend.h1.gender.female+100 ON 0 FROM patients
```

```
                                 Expression
                                        626
```

As noted earlier, the value of a member expression depends upon the measure that is in use:

```
SELECT gend.h1.gender.female ON 0 FROM patients WHERE MEASURES.[avg age]
```

```
                                    Female
                                      37.23
```

Similarly:

```
SELECT gend.h1.gender.female+500 ON 0 FROM patients WHERE MEASURES.[avg age]
```

```
                                 Expression
                                      537.23
```

The next example shows the numeric value of a dimension:

```
SELECT allerd ON 0 FROM patients
```

```
                                      1,000
```

As noted earlier, the value of such an expression depends upon the measure that is in use:

```
SELECT allerd ON 0 FROM patients WHERE MEASURES.[avg allergy count]
```

```
                                       0.99
```

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 an MDX function to return a string. These functions include:
  - PROPERTIES, which returns 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.
  - IIF, which returns one of two values, depending on the value of a given logical expression.
  - LOOKUP, which looks up a given key in a term list and returns a substitute string.
  - %LOOKUP, which returns one value from a term list.
- 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")
  ```
- A reference to a pivot variable that contains a string value. To refer to a pivot variable, use the following syntax:

  ```
  $VARIABLE.variablename
  ```

  Where *variablename* is the logical variable name. Do not enclose this expression with square brackets. This syntax is not case-sensitive; nor is the pivot variable name.

  For information on defining pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

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]`:

  ```
  MEASURES.[%COUNT]>1100
  ```

  For another example, the following expression returns true or false, depending on the name of the current member:

  ```
  colord.CURRENTMEMBER.PROPERTIES("Name")="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]`:

  ```
  (MEASURES.[%COUNT]>1100) AND (MEASURES.[%COUNT]<1500)
  ```

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

If	Then
Both expressions are numeric	Perform a numeric comparison
Both expressions are strings	Perform a string comparison
One expression is numeric and the other is a string	The numeric expression is less than the string expression

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

```
homed.CURRENTMEMBER = homed.h1.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:

```
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 expression can have either of the following forms:

- A tuple literal, which is a comma-separated list of member expressions, enclosed by parentheses:

  ```
  (member_expr1, member_expr2, member_expr3, ...)
  ```

  Where *member_expr1*, *member_expr2*, *member_expr3*, and so on are member expressions. The parentheses are required.

- A reference to a pivot variable that contains a tuple literal. To refer to a pivot variable, use the following syntax:

  ```
  $VARIABLE.variablename
  ```

  Where *variablename* is the logical variable name. Do not enclose this expression with square brackets. This syntax is not case-sensitive; nor is the pivot variable name.

  For information on defining pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

Notes and Additional Terminology

In other implementations of MDX, 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 `(allerd.soy,allerd.wheat)` represents all patients who are allergic to both soy and wheat, and this tuple could potentially have a non-null value.

If the tuple refers to 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:

```
(allerd.[dairy products], colord.red, aged.35)
```

Another partially qualified tuple is as follows:

```
(diagd.asthma, aged.[age group].[30 to 59], MEASURES.[%COUNT])
```

Also, note that each data cell 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:

```
[dimension_name].[hierarchy_name].[level_name].[member_name]
```

  Where:

  – [*dimension_name*] is an MDX identifier that names a dimension.

  – [*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.

    You can also omit the hierarchy name when you refer to a calculated member, because a calculated member is not in a hierarchy.

  – [*level_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.

    You can also omit the level name when you refer to a calculated member, because a calculated member is not in a level.

  – [*member_name*] is an MDX identifier that names a member of that level.

  For example:

```
[gend].[h1].[gender].[female]
```

  For a measure, a member literal has the following variant form, because any measure is a member of a dimension called Measures. This dimension does not have a hierarchy or a level.

```
[MEASURES].[measure_name]
```

  For example:

```
[MEASURES].[%COUNT]
```

  Note that although a measure is a member, an expression like the preceding returns a scalar value and cannot be used in all the same ways as other member expressions.

- An explicit reference to a hierarchy as follows:

```
[dimension_name].[hierarchy_name]
```

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

  For example:

```
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:

```
[gend].[h1].[gender].female.NEXTMEMBER
```

  Many MDX functions return members, including LAG, NEXTMEMBER, PARENT, and others.

  Note that %TERMLIST can return a member.

- An expression that uses the internal key of a member to return that member, via the following syntax:

```
[dimension_name].[hierarchy_name].[level_name].&[member_key]
```

  Where *member_key* is the value of the KEY property of the member. For example:

```
birthqd.h1.quarter.&[2]
```

  Note that *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 KEY values are created, see the reference "Intrinsic Properties." Also note that you can use the PROPERTIES function to find the value of the 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:

```
[relationship_name].member_expression
```

  Where [*relationship_name*] is an MDX identifier that names a relationship in the cube used by the query and *member_expression* is a member expression suitable for that cube.

- A reference to a pivot variable that contains a member expression. To refer to a pivot variable, use the following syntax:

```
$VARIABLE.variablename
```

  Where *variablename* is the logical variable name. Do not enclose this expression with square brackets. This syntax is not case-sensitive; nor is the pivot variable name.

  For information on defining pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

For example, the following member expressions are all equivalent:

```
[gend].[h1].[gender].Female
[gend].female
gend.H1.gender.female
gend.h1.FEMALE
gend.female
```

Calculated Members

Members can be defined within the cube definition, as part of the definition of a level. You can also create *calculated members*, which are typically based on other members. You can define calculated members in two ways:

- Within the WITH clause of a query. The member is available within the rest of the query, but is not available in other queries.
- Within the 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 WHERE clause.
- As an element of a 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:

```
[dimension_name].[hierarchy_name].[level_name]
```

  Where:

  – [*dimension_name*] is an MDX identifier that names a dimension.

  – [*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.

  – [*level_name*] is an MDX identifier that names a level within that hierarchy.

  For example:

```
[gend].[h1].[gender]
```

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

```
[relationship_name].level_expression
```

  Where *relationship_name* is the name of a relationship in the cube used by the query and *level_expression* refers to a level contained in that relationship.

Uses

You can use level expressions as arguments to the following MDX functions:

- %TOPMEMBERS
- ALLMEMBERS
- MEMBERS
- ANCESTOR
- CLOSINGPERIOD
- OPENINGPERIOD
- PARALLEL PERIOD

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:

```
[dimension_name].[hierarchy_name]
```

  Where:

  – [*dimension_name*] is an MDX identifier that names a dimension.

  – [*hierarchy_name*] is an MDX identifier that names a hierarchy within that dimension.

  For example:

```
[gend].[h1]
```

- A reference to a dimension:

```
[dimension_name]
```

  For example:

```
[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:

```
[relationship_name].hierarchy_expression
```

  Where *relationship_name* is the name of a relationship in the cube used by the query and *hierarchy_expression* refers to a hierarchy contained in that relationship.

Uses

A bare hierarchy expression returns all records; this expression is equivalent to an All member. (Note that you can use a bare hierarchy expression even if the dimension does not formally define an All level.) For example:

```
SELECT MEASURES.[%count] ON 0, colord.h1 ON 1 FROM patients
```

Patient Count
1,000

Also, you can use a hierarchy expression as an argument to any of the following functions:

- %TOPMEMBERS
- ALLMEMBERS
- CURRENTMEMBER
- 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.

  Note that %TERMLIST can return a set.

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

  (Note that in other implementations of MDX, for each 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.)

- A reference to a pivot variable that contains a set expression. To refer to a pivot variable, use the following syntax:

  ```
  $VARIABLE.variablename
  ```

  Where *variablename* is the logical variable name. Do not enclose this expression with square brackets. This syntax is not case-sensitive; nor is the pivot variable name.

  For information on defining pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

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 an InterSystems extension to MDX.

Details

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

```
%SEARCH.&[comparison expression]
```

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

```
[MEASURES].[test score]>60
```

Note: Both sets of square brackets are required: the square brackets around the comparison expression *and* the square brackets of the measure identifier in the comparison expression. Hence a valid search expression always starts with `%SEARCH.&[[`

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

```
SELECT FROM patients WHERE %SEARCH.&[[MEASURES].[Test Score]>60]
Result:                6,191
```

More generally, *comparison expression* can be a combination of logical expressions. This expression can include:

- Logical comparison operators: > (greater than), >= (greater than or equal to), = (equal to), < (less than), and <= (less than or equal to).

  If the searchable measure contains string values, you can also use the SQL LIKE operator.
- The AND operator, the OR operator, and parentheses to control precedence.
- Numeric literals.
- String literals enclosed in single quotes.
- The SQL expressions IS NULL and IS NOT NULL. For example:

  ```
  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 an InterSystems 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]
```

Or:

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

Where:

- *catalog* is the catalog to which the quality measure belongs.
- *set* is a set in that catalog.
- *qm name* is the short name of the quality measure. (The full name of a quality measure is *catalog/set/qm name*.)
- *group name* is the name of a group defined in the given quality measure.

The first expression returns the value of the quality measure. The second expression returns the value of the given group.

Uses

You can use quality measure expressions in the same way that you use other measures.

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 an InterSystems extension to MDX.

Syntax and Details

```
select_statement %FILTER 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 %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 to you 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. This is the process of *axis folding* (a filter is considered to be a query axis). Axis folding means that if a given source record has a non-null result for each slicer axis, that record is counted multiple times.

In axis folding, values are combined according to the aggregation method for that measure, as specified in the cube definition. (In the examples here, `%COUNT` is added.)

For more details, see "Axis Folding" in the appendix "How the DeepSee Query Engine Works" in the *DeepSee Implementation Guide*.

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 an InterSystems extension to MDX that considers the measure values in the fact table itself.)

```
SELECT MEASURES.[%COUNT] ON 0 FROM patients %FILTER %SEARCH.&[[MEASURES].[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

```
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 both axes. 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:

```
>>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:

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

```
DRILLFACTS select_statement
```

Or:

```
DRILLFACTS select_statement RETURN fieldname1, fieldname2, ...
```

Or:

```
DRILLFACTS 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 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 optionalmaxrows select_statement
```

Or:

```
DRILLTHROUGH optionalmaxrows select_statement RETURN fieldname1, fieldname2, ...
```

Or:

```
DRILLTHROUGH optionalmaxrows select_statement RETURN fieldname1, ... %ORDER BY
fieldname3, ...
```

Or:

```
DRILLTHROUGH optionalmaxrows select_statement %LISTING [listingname]
```

Where:

- *optionalmaxrows*, which is optional, has the form `MAXROWS` *integer*

  This argument specifies the maximum number of rows to return. The default is 1000.

- *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.

- *listingname* is the name of a detail listing. This listing must already be defined and the user must have permission to use it. If this name does not include spaces, you can omit the square brackets around it. The name of the listing is case-sensitive.

The RETURN clause specifies the fields to display. The %ORDER BY clause is an InterSystems extension to MDX. If included, this clause specifies how to sort the displayed records.

The %LISTING clause is another InterSystems extension to MDX. This clause specifies the listing to use. If you specify this clause, DeepSee displays the fields as given in that listing.

If you do not specify the RETURN clause or the %LISTING clause, the query uses the default listing defined in the DeepSee cube.

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 "Implicit Joins (Arrow Syntax)" 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:

```
... 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   DoctorGrou
   1: SUBJ_10161 0          F          76         3          I
   2: SUBJ_10330 0          F                     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 MAXROWS argument:

```
DRILLTHROUGH MAXROWS 3 SELECT homed.Magnolia ON 1 FROM patients

   #  PatientID  Age        Gender     TestScore  HomeCity   DoctorGrou
   1: SUBJ_10161 0          F          76         3          I
   2: SUBJ_10330 0          F                     3          II
   3: SUBJ_10554 0          F          68         3          II
```

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_101616    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_101616    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 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

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

---------------------------------------------------------------------------
Elapsed time:       .011952s
```

See Also

- CREATE MEMBER Statement

DROP SET Statement

Removes a named set defined earlier in the current session.

Syntax and Details

```
DROP SET cube_name.set_name
```

- *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

```
>>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:

Base Unit	Meaning	Example
#	Display the value without the thousands separator. Do not include any decimal places.	12345
#,#	Display the value with the thousands separator. Do not include any decimal places. This is the default display format for positive numbers.	12,345
#.##	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.	12345.67
#,#.##	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.	12,345.67

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. (This clause is relevant only if the query contains calculated members on both axes.)

Examples

The following table shows some examples:

Example	Logical Value	Display Value
`FORMAT_STRING='#,#;(#,#);'` Note that this corresponds to the default way in which numbers are displayed.	6608.9431	6,609
	–1234	(1234)
`FORMAT_STRING='#,#.###;'`	6608.9431	6,608.943
`FORMAT_STRING='#%;'`	6	600%
`FORMAT_STRING='$#,#;($#,#);'`	2195765	$2,195,765
	–3407228	($3,407,228)

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

```
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:

  ```
  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, which is equivalent) or `ROWS` (or 1, which is equivalent).

  You can specify zero, one, or two ON clauses. If you specify two ON clauses, you can specify them in either order.

  In other implementations of MDX, if you specify `ROWS`, you must also specify `COLUMNS`. In DeepSee MDX, if you specify `ROWS` but do not specify `COLUMNS`, DeepSee automatically generates an ON clause for `COLUMNS`. This clause uses the count measure.

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

If you omit the ON clause, 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.

For any given cell of the results, to determine the value to use, DeepSee finds the *intersection* of the member used for the column and the member (if any used) for the row:

- If one member is a measure and the other is not a measure, DeepSee finds the value of that measure for that member. For example, if one member is the `Ave Age` measure, and the other member is the `34577` ZIP code, then the corresponding data cell contains the average age of patients whose home ZIP code is 34577.
- If neither member is a measure, DeepSee uses the default measure, which is usually `%COUNT`. For example, if one member is the gender `F`, and the other member is the `34577` ZIP code, then the corresponding data cell contains the count of all female patients whose home ZIP code is 34577.

Example

The following simple example shows patient counts by ZIP code.

```
SELECT MEASURES.[%COUNT] ON 0, homed.zip.MEMBERS ON 1 FROM patients
                              Patient Count
1 32006                                2,272
2 32007                                1,111
3 34577                                3,399
4 36711                                1,069
5 38928                                2,149
```

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:

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

	Patient Count
1 Cedar Falls	1,097
2 Centerville	1,136
3 Cypress	1,124
4 Elm Heights	1,089
5 Juniper	1,133
6 Magnolia	1,063
7 Pine	1,124
8 Redwood	1,083
9 Spruce	1,151

See Also

- DRILLTHROUGH Statement
- WHERE Clause
- WITH Clause

SET Statement

Creates a pivot variable for use in the current session, for the purpose of development. This statement is available only in the MDX shell.

Syntax and Details

```
SET variable_name variable_value
```

Where:

- *variable_name* is the name of the pivot variable (without the leading `$variable.` syntax). The name is not case-sensitive; DeepSee ignores the case when you refer to the variable. You cannot create multiple variables with names that differ only in case.

  Note that this pivot variable is independent of any cube and is available only for use within the current shell session.

- *variable_value* is the value of the pivot variable.

  For information on pivot variables, see "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

Examples

The following MDX shell session provides an example:

```
>>set mypivotvar colord.red
mypivotvar is: colord.red
>>select $variable.mypivotvar on 0 from patients

                                      Red
                                       129
```

SOLVE_ORDER Clause

Used with a definition of a calculated member, this clause specifies the order in which to apply the definition of this calculated member relative to other calculated members. This clause is relevant only if the query contains calculated members on both 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.

```
SOLVE_ORDER=integer
```

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

This clause is useful in a query that has a calculated member as a column and a conflicting calculated member as a row. This clause affects how the system determines the value in the cell *and* the format applied to the cell. By default, the *column* determines FORMAT_STRING.

If you instead want the row to determine the format and the value, ensure that SOLVE_ORDER is higher for the calculated member used as a row.

With one exception, the calculated member with the higher SOLVE_ORDER is evaluated last and thus controls the results. The exception is that if a row or column uses the %CELL function, its implicit default SOLVE_ORDER is 10.

If the row and column both use %CELL and you want the row to determine the value and format string, set the SOLVE_ORDER to 11 for the row.

If the column and row members have the same SOLVE_ORDER, the column member controls the results, as in the default case.

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

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 (of all %FILTER clauses) are aggregated together. This is the process of *axis folding* (a filter is considered to be a query axis). Axis folding means that if a given source record has a non-null result for each slicer axis, that record is counted multiple times.

In axis folding, values are combined according to the aggregation method for that measure, as specified in the cube definition. (In the examples here, `%COUNT` is added.)

For more details, see "Axis Folding" in the appendix "How the DeepSee Query Engine Works" in the *DeepSee Implementation Guide*.

Example

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

```
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:

```
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:

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

	Patient Count
1 Cedar Falls	137
2 Centerville	129
3 Cypress	150
4 Elm Heights	128
5 Juniper	126
6 Magnolia	143
7 Pine	155
8 Redwood	148
9 Spruce	147

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}
```

	Patient Count
1 Cedar Falls	646
2 Centerville	698
3 Cypress	667
4 Elm Heights	659
5 Juniper	700
6 Magnolia	670
7 Pine	724
8 Redwood	701
9 Spruce	704

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)
```

	Patient Count
1 Cedar Falls	56
2 Centerville	65
3 Cypress	80
4 Elm Heights	59
5 Juniper	73
6 Magnolia	74
7 Pine	82
8 Redwood	70
9 Spruce	74

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)
```

	Patient Count
1 Cedar Falls	3
2 Centerville	9
3 Cypress	7
4 Elm Heights	1
5 Juniper	8
6 Magnolia	2
7 Pine	5
8 Redwood	6
9 Spruce	3

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]
```

	Female	Male
All Patients	74.78	74.46

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 an InterSystems 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 calculated members, named sets, or parameters 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:

```
%OR({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" 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 both axes. 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
```

```
                              Patient Count
1 Cedar Falls                         1,045
2 Centerville                         1,069
3 Cypress                             1,150
4 Elm Heights                         1,104
5 Juniper                             1,155
6 Magnolia                            1,111
7 Pine                                1,138
8 Redwood                             1,111
9 Spruce                              1,117
```

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 an InterSystems 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:

```
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
```

```
                        Patient Count         pct age grps
1 0 to 29                       4,216                 0.42
2 30 to 59                      4,212                 0.42
3 60+                           1,572                 0.16
```

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:

```
WITH MEMBER MEASURES.[BADpct age grps] AS 'aged.CURRENTMEMBER/aged.[all patients]', FORMAT_STRING='#.##'

SELECT {MEASURES.[%COUNT],MEASURES.[BADpct age grps]} ON 0,
aged.h1.[age group].MEMBERS ON 1 FROM patients
```

```
                        Patient Count      BADpct age grps
1 0 to 29                       4,216                 1.00
2 30 to 59                      4,212                 1.00
3 60+                           1,572                 1.00
```

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.h1.[age group].MEMBERS ON 1 FROM patients
```

	Patient Coun	pct age grps	Patient Coun	pct age grps
1 0 to 29	1,985	0.39	2,231	0.45
2 30 to 59	2,123	0.42	2,089	0.42
3 60+	926	0.18	646	0.13

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.h1.[age group].MEMBERS ON 1 FROM patients
WHERE aged.h1.[age group].[0 to 29]
```

	Patient Count	pct of all ages
1 0 to 29	4,216	1.00
2 30 to 59	*	*
3 60+	*	*

See Also

- %MDX

%CELL

Returns the value of another cell in a pivot table, by position. This function is an InterSystems 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 returns the value of the given cell, or null if the cell has no value.

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
```

```
                     Rainfall Inches           Expression
 1 1960                       177.83                177.83
 2 1961                       173.42                351.25
 3 1962                       168.11                519.36
 4 1963                       188.30                707.66
 5 1964                       167.58                875.24
 6 1965                       175.23              1,050.47
 7 1966                       182.50              1,232.97
 8 1967                       154.44              1,387.41
 9 1968                       163.97              1,551.38
10 1969                       184.84              1,736.22
11 1970                       178.31              1,914.53
```

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
```

```
                     Rainfall Inches    Cumulative Inches
 1 1960                       177.83                177.83
 2 1961                       173.42                351.25
 3 1962                       168.11                519.36
 4 1963                       188.30                707.66
 5 1964                       167.58                875.24
 6 1965                       175.23              1,050.47
 7 1966                       182.50              1,232.97
 8 1967                       154.44              1,387.41
 9 1968                       163.97              1,551.38
10 1969                       184.84              1,736.22
11 1970                       178.31              1,914.53
```

See Also

- %CELLZERO

%CELLZERO

Returns the value of another cell in a pivot table or returns zero if that cell has no value. This function is a DeepSee extension to MDX.

Returned Type

This function returns a number or a string.

Syntax and Details

```
%CELLZERO(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 returns the value of the given cell, or zero if the cell has no value.

Example

The following query uses three measures. The first measure displays the number of units sold in a given period. The second measure displays the number of units sold in the previous period; this is a calculated measure defined with PREVMEMBER. The third measure displays the change in units sold since the previous period; this is a calculated measure defined with %CELLZERO.

```
WITH  MEMBER [MEASURES].[UnitsSoldPreviousPeriod]
AS '%LABEL(([DateOfsale].[Actual].CurrentMember.PrevMember,MEASURES.[units sold]),"Units (Prev Period)")'

MEMBER [MEASURES].[Delta Since Prev Period] AS '%CELLZERO(-2,0)-%CELLZERO(-1,0)'
SELECT
{[Measures].[Units Sold],[MEASURES].[UNITSSOLDPREVIOUSPERIOD],[MEASURES].[DELTA SINCE PREV PERIOD]} ON
 0,
[DateOfSale].[Actual].[MonthSold].Members ON 1
FROM [HoleFoods]
%FILTER [PRODUCT].[P1].[PRODUCT CATEGORY].&[Dairy]
```

```
                        Units Sold   Units (Prev Peri   Delta Since Prev
 1 Jan-2009                      *                  *                  0
 2 Feb-2009                      *                  *                  0
 3 Mar-2009                      *                  *                  0
 4 Apr-2009                      1                  *                  1
 5 May-2009                      *                  1                 -1
 6 Jun-2009                      8                  *                  8
 7 Jul-2009                      1                  8                 -7
 8 Aug-2009                      *                  1                 -1
 9 Sep-2009                      *                  *                  0
10 Oct-2009                      *                  *                  0
11 Nov-2009                      *                  *                  0
12 Dec-2009                      *                  *                  0
13 Jan-2010                      1                  *                  1
14 Feb-2010                      *                  1                 -1
15 Mar-2010                      2                  *                  2
...
```

See Also

- %CELL

%FIRST

Returns the value of the given measure (or other numeric expression) evaluated for the first non-empty member of a set. This function is an InterSystems extension to MDX.

Returned Type

This function returns a number.

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.

Example

For reference, the following query shows patients with asthma, grouped by birth decade:

```
SELECT MEASURES.[%Count] ON 0, birthd.decade.MEMBERS ON 1 FROM patients WHERE diagd.asthma
```

```
                            Patient Count
 1 1910s                                *
 2 1920s                                *
 3 1930s                                1
 4 1940s                                9
 5 1950s                                8
 6 1960s                               11
 7 1970s                               12
 8 1980s                               14
 9 1990s                               11
10 2000s                               14
11 2010s                                4
```

The following query uses %FIRST to get the first non-empty set of patients from the preceding set:

```
SELECT MEASURES.[%Count] ON 0, %FIRST(birthd.decade.MEMBERS) ON 1 FROM patients WHERE diagd.asthma
```

```
                            Patient Count
FIRST                                  1
```

See Also

- %LAST

%KPI

Returns a value from a KPI or plugin. This function is an InterSystems 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 or plugin.
- *propName* is the quoted name of a `<property>` element of the 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.
- *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.

  You can specify up to 16 parameters and their values.
- *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"`. (The %MDX function also uses this parameter; see %MDX for examples.)

Important: If you use the *%CONTEXT* parameter, remember to enclose it in quotes. Also, you must specify a value for this parameter unless you use it as the last parameter. For example, the following is valid:

```
%KPI("%DeepSee.Median","MEDIAN",1,"%measure","Amount Sold","%CONTEXT")
```

The following is also valid:

```
%KPI("%DeepSee.Median","MEDIAN",1,"%CONTEXT","all","%measure","Amount Sold")
```

But the following is *not* correct and will not be interpreted as desired:

```
%KPI("%DeepSee.Median","MEDIAN",1,"%CONTEXT","%measure","Amount Sold")
```

KPIs (other than plugins) are executed synchronously by default but can be defined to execute asynchronously.

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]
```

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

Available Plugin Classes

DeepSee provides several plugins for you to use with the %KPI function:

%DeepSee.Distinct

Gets the number of distinct values for the given level.

This plugin provides the property `DISTINCT`. The plugin accepts the following parameters:

Parameter	Value
`%cube`	Logical name of the cube.
`%level`	MDX identifier for the level or relationship whose distinct values you want to count. For example, `[DocD].[H1].[Doctor]` or `[RelatedCubes/Doctors].[DocD]`

This plugin is defined by the class %DeepSee.PlugIn.Distinct.

%DeepSee.Median

Gets the median value for a given measure, across all the lowest-level records used in a cell.

This plugin provides the property `MEDIAN`. The plugin accepts the following parameters:

Parameter	Value
%cube	Logical name of the cube.
%measure	MDX identifier for measure whose values you want to use. For example: [MEASURES].[Measure Name]

This plugin is defined by the class %DeepSee.PlugIn.Median.

%DeepSee.Percentile

Gets a percentile value for a given measure, across all the lowest-level records.

This plugin provides the property PERCENTILE. The plugin accepts the following parameters:

Parameter	Value
%cube	Logical name of the cube.
%measure	MDX identifier for measure whose values you want to use. For example: [MEASURES].[Measure Name]
%percentile	The percentile that you want to evaluate. The default is 50, so that the plugin calculates the 50th percentile.

This plugin is defined by the class %DeepSee.PlugIn.Percentile.

Note that these plugin classes are defined with *PLUGINTYPE* as "Aggregate", which means that the plugins cannot be directly used in the Analyzer or in widgets.

See Also

- %MDX

%LABEL

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
```

```
                                  my label
1 None                                 1.04
2 Blue                                 1.01
3 Green                                1.02
4 Orange                               1.01
5 Purple                               1.06
6 Red                                  1.03
7 Yellow                               1.00
```

In contrast:

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

```
                          Avg Allergy Count
1 None                                 1.04
2 Blue                                 1.01
3 Green                                1.02
4 Orange                               1.01
5 Purple                               1.06
6 Red                                  1.03
7 Yellow                               1.00
```

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

See Also

- IIF
- ISNULL

%LAST

Returns the value of the given measure (or other numeric expression) evaluated for the last non-empty member of a set. This function is an InterSystems extension to MDX.

Returned Type

This function returns a number.

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.

Example

For reference, the following query shows patients with osteoporosis, grouped by birth decade:

```
SELECT MEASURES.[%Count] ON 0, birthd.decade.MEMBERS ON 1 FROM patients WHERE diagd.osteoporosis
```

```
                            Patient Count
 1 1910s                                2
 2 1920s                                5
 3 1930s                               10
 4 1940s                                5
 5 1950s                                *
 6 1960s                                *
 7 1970s                                *
 8 1980s                                *
 9 1990s                                *
10 2000s                                *
11 2010s                                *
```

The following query uses %LAST to get the last non-empty set of patients from the preceding set:

```
SELECT MEASURES.[%Count] ON 0, %LAST(birthd.decade.MEMBERS) ON 1 FROM patients WHERE diagd.osteoporosis
```

```
                            Patient Count
LAST                                    5
```

See Also

- %FIRST

%LIST

Returns a comma-separated list of values, given a set of values. This function is an InterSystems 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
```

	Patient Count
1 None	8,603
2 asthma	676
3 CHD	345
4 diabetes	546
5 osteoporosis	212
6 LIST	8603,676,345,546,212

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 Scorecard with Trend Lines dashboard.

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

	Patient Count	LIST
1 Cedar Falls	1,079	11,22,42,92,115,159,176,135,127,179,21
2 Centerville	1,058	3,24,51,57,114,157,167,152,161,145,27
3 Cypress	1,095	12,14,57,84,105,146,160,159,153,166,39
4 Elm Heights	1,086	8,22,54,72,121,154,168,135,176,143,33
5 Juniper	1,152	9,21,56,79,124,165,200,152,169,155,22
6 Magnolia	1,152	6,18,55,83,116,181,190,159,172,145,27
7 Pine	1,120	8,22,47,72,108,160,166,173,161,170,33
8 Redwood	1,132	6,17,61,90,109,174,189,129,171,154,32
9 Spruce	1,126	11,20,70,74,107,167,171,161,164,156,25

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 or trend bar graphic.

%LOOKUP

Returns one value from a term list. This function is an InterSystems extension to MDX.

Returned Type

This function returns a number or a string.

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.

For information on defining term lists, see "Defining Term Lists" in the *Advanced DeepSee Modeling Guide*.

Comparison with Other Term List Functions

The following table compares functions that you can use with term lists:

Function	Purpose	Return Value
%LOOKUP	Looks up a value, given the key of a term list item. Returns the value of the term list item. You can specify a default value to return.	A number or a string (which could be the name of a member).
LOOKUP	Returns a field from a term list item. By default, this field is the key field, but you can return another field instead. You can specify a default value to return.	
%TERMLIST	Returns a set based on the given term list.	Returns a set.

Example

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

```
key      value
CutOff   10000000
```

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:

```
SELECT
FILTER(Outlet.City.Members,Outlet.H1.City.CurrentMember.Properties("Population")>%LOOKUP("Values","CutOff"))

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. These must be quoted.

  The order in which you list parameters does not affect the query.

  You can specify up to 16 parameters and their values.

- *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.

Important: Remember to quote the *%CONTEXT* parameter if you use it.

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
```

```
                        Patient Count             FRACTION
1 None                          8,428                 0.84
2 asthma                          712                 0.07
3 CHD                             343                 0.03
4 diabetes                        485                 0.05
5 osteoporosis                    212                 0.02
```

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
```

```
                                    NAME
 1 Amsterdam                        1,633
 2 Antwerp                            421
 3 Atlanta                          3,331
 4 Bangalore                        3,786
...
```

In this case, the subquery is as follows:

```
WITH %PARM City AS 'value:[All Cities]' SELECT FROM HOLEFOODS WHERE Outlet.@City
```

The following examples show the effect of the %CONTEXT parameter. First, the following query uses %MDX without *%CONTEXT*:

```
WITH
MEMBER [MEASURES].[PercentOfAllRevenue] AS '100 * MEASURES.[Amount Sold] /
%MDX("SELECT MEASURES.[Amount Sold] ON 0 FROM holefoods")'

SELECT NON EMPTY [Channel].[H1].[Channel Name].Members ON 0,
NON EMPTY [Product].[P1].[Product Category].Members ON 1
FROM [HoleFoods]
WHERE [MEASURES].[PERCENTOFALLREVENUE]
```

	No Channel	Online	Retail
1 Candy	0.28	0.83	0.61
2 Cereal	0.11	0.58	0.39
3 Dairy	0.23	2.43	1.01
4 Fruit	1.04	7.55	3.76
5 Pasta	0.79	7.19	4.14
6 Seafood	3.60	22.23	10.41
7 Snack	2.58	10.84	7.28
8 Vegetable	1.42	6.63	4.05

The calculated member `[MEASURES].[PercentOfAllRevenue]` computes the Amount Sold measure for the current pivot table cell, divided by the aggregate value for that measure across the entire cube. This value is then divided by 100, so the values displayed in the results add up to 100.

In contrast, consider the results when we use *%CONTEXT* as `"rows"`:

```
WITH
MEMBER [MEASURES].[PercentOfRows] AS '100 * MEASURES.[Amount Sold] /
%MDX("SELECT MEASURES.[Amount Sold] ON 0 FROM holefoods","%CONTEXT","rows")'

SELECT NON EMPTY [Channel].[H1].[Channel Name].Members ON 0,
NON EMPTY [Product].[P1].[Product Category].Members ON 1
FROM [HoleFoods]
WHERE [MEASURES].[PercentOfRows]
```

	No Channel	Online	Retail
1 Candy	16.08	48.30	35.62
2 Cereal	10.29	53.69	36.02
3 Dairy	6.38	66.15	27.48
4 Fruit	8.41	61.14	30.45
5 Pasta	6.49	59.33	34.18
6 Seafood	9.93	61.34	28.73
7 Snack	12.46	52.38	35.16
8 Vegetable	11.72	54.77	33.51

In this case, the %MDX subquery uses the row context. As a result, the numbers in each row add up to 100.

Now consider the results when we use *%CONTEXT* as `"columns"`:

```
WITH
MEMBER [MEASURES].[PercentOfCols] AS '100 * MEASURES.[Amount Sold] /
%MDX("SELECT MEASURES.[Amount Sold] ON 0 FROM holefoods","%CONTEXT","columns")'

SELECT NON EMPTY [Channel].[H1].[Channel Name].Members ON 0,
NON EMPTY [Product].[P1].[Product Category].Members ON 1
FROM [HoleFoods]
WHERE [MEASURES].[PercentOfCols]
```

	No Channel	Online	Retail
1 Candy	2.76	1.43	1.94
2 Cereal	1.10	0.99	1.22
3 Dairy	2.34	4.18	3.19
4 Fruit	10.35	12.96	11.88
5 Pasta	7.83	12.34	13.08
6 Seafood	35.83	38.14	32.89
7 Snack	25.67	18.60	22.99
8 Vegetable	14.12	11.37	12.80

In this case, the numbers in each column add up to 100.

See Also

- %KPI

%NOT

Enables you to exclude a single member of a given level. This function is an InterSystems extension to MDX.

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, first consider the following query, which uses EXCEPT:

```
SELECT aged.[age bucket].MEMBERS ON 1 FROM patients WHERE EXCEPT(aged.[age group].MEMBERS,aged.[age
group].[0 to 29])
```

```
1 0 to 9                                  *
2 10 to 19                                *
3 20 to 29                                *
4 30 to 39                              166
5 40 to 49                              139
6 50 to 59                              106
7 60 to 69                               86
8 70 to 79                               62
9 80+                                    41
```

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

```
SELECT aged.[age bucket].MEMBERS ON 1 FROM patients WHERE aged.[age group].[0 to 29].%NOT
```

```
 1 0 to 9                                  *
2 10 to 19                                *
3 20 to 29                                *
4 30 to 39                              166
5 40 to 49                              139
6 50 to 59                              106
7 60 to 69                               86
8 70 to 79                               62
9 80+                                    41
```

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

```
SELECT aged.[age group].[0 to 29].%NOT ON 1 FROM patients
```

```
Not 0 to 29                             600
```

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 into a single member, for efficiency and to avoid double-counting. This function is an InterSystems extension to MDX.

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 or tuples. This expression must be enclosed in curly braces.

This function enables you to combine the given members or tuples 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}


1 Female                                  56
2 Male                                    59
```

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})


1 Female                                  55
2 Male                                    57
```

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

As of release 2014.1, you can use %OR with a set that contains members of different levels (or even that contains tuples). For example:

```
SELECT NON EMPTY [Measures].[%COUNT] ON 0 FROM [Patients]
WHERE %OR({[AgeD].[H1].[Age Bucket].&[80+],[DiagD].[H1].[Diagnoses].&[CHD]})

                             Patient Count
                                        71
```

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.

Also see examples in "Defining Calculated Members" in *Defining DeepSee Models*.

%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 an InterSystems extension to MDX.

Returned Type

This function returns an empty string.

Syntax and Details

```
%SPACE()
```

Example

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

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

%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 an InterSystems extension to MDX.

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 *Advanced DeepSee Modeling Guide*.

Example

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

Terms

key	value
Atlanta	Atlanta
Boston	Boston
New York	New York

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 (which contains "Comparison with Other Term List Functions")
- LOOKUP
- %OR

%TIMERANGE

Enables you to define a range of time members, possibly open-ended. This function is an InterSystems extension to MDX.

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.

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 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])

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

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.[jan 05 1924])

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

%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 an InterSystems 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:

- *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 is equivalent to the 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 **Rows** or **Columns**. Specifically, when you drag and drop a dimension, the Analyzer uses the expression `[dimension_name].[hierarchy_name].%TOPMMEMBERS`, where *hierarchy_name* is the first hierarchy defined in the dimension.

Example

For example, consider the following cube contents:

```
BirthD
  H1
    Decade
    Year
    Period
    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
                    Patient Count    Encounter Count     Avg Test Score
 1 1910s                       80              5,359              75.17
 2 1920s                      227             12,910              74.20
 3 1930s                      567             33,211              74.67
 4 1940s                      724             38,420              73.39
 5 1950s                    1,079             46,883              73.72
 6 1960s                    1,475             57,814              74.16
 7 1970s                    1,549             49,794              74.35
 8 1980s                    1,333             35,919              74.13
 9 1990s                    1,426             29,219              74.79
10 2000s                    1,406             20,072              74.95
11 2010s                      134              1,346              73.55
```

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
                    Patient Count    Encounter Count     Avg Test Score
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. Also see examples in "Defining Calculated Members" in *Defining DeepSee Models*.

See Also

- AVG

- 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:

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

```
                            Patient Count
1 32006                             2,272
2 32007                             1,111
3 34577                             3,399
4 36711                             1,069
5 38928                             2,149
```

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

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

```
SELECT MEASURES.ALLMEMBERS ON 1, gend.gender.MEMBERS on 0 FROM patients
                             Female                 Male
1 Patient Count                5,067                 4,933
2 Age                        187,139               170,117
3 Avg Age                      36.93                 34.49
4 Allergy Count                3,067                 3,131
5 Avg Allergy Count             1.02                  1.04
6 Encounter Count            169,164               158,183
7 Avg Encounter Cou            33.39                 32.07
8 Test Score                 302,267               298,818
9 Avg Test Score               74.78                 74.46
```

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

```
ANCESTOR(member_expression,ancestor_level)
```

Where:

- *member_expression* is an expression that returns a member.

  This expression cannot refer to a measure.

- *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.

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:

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

```
                          Patient Count
1943                                 76
```

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

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

```
                          Patient Count
Mar-1943                             5
```

See Also

- CHILDREN
- CLOSINGPERIOD
- COUSIN
- DESCENDANTS
- 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

```
AVG(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 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 *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:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
birthd.decade.MEMBERS ON 1 FROM patients
                    Patient Count    Encounter Count     Avg Test Score
 1 1910s                       80              5,359              75.17
 2 1920s                      227             12,910              74.20
 3 1930s                      567             33,211              74.67
 4 1940s                      724             38,420              73.39
 5 1950s                    1,079             46,883              73.72
 6 1960s                    1,475             57,814              74.16
 7 1970s                    1,549             49,794              74.35
 8 1980s                    1,333             35,919              74.13
 9 1990s                    1,426             29,219              74.79
10 2000s                    1,406             20,072              74.95
11 2010s                      134              1,346              73.55
```

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,
AVG(birthd.decade.MEMBERS) ON 1 FROM patients
                    Patient Count    Encounter Count     Avg Test Score
AVG                        909.09          30,086.09              74.28
```

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:

```
SELECT AVG(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
                                        AVG
                                     909.09
```

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

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
                              Patient Count
1 1910s                                71
2 2010s                               155
3 1920s                               223
```

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
```

```
                              Patient Count
 1 1910s                                  6
 2 1920s                                 13
 3 2010s                                 44
 4 1940s                                 54
 5 1930s                                 56
 6 1950s                                107
 7 1970s                                128
 8 1960s                                136
 9 1990s                                144
10 1980s                                155
11 2000s                                157
```

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
```

```
                              Patient Count
1 1910s                                  6
2 1920s                                 13
3 2010s                                 44
4 1940s                                 54
5 1930s                                 56
6 1950s                                107
7 1970s                                128
```

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

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
                              Patient Count
1 1910s                                71
2 2010s                               155
3 1920s                               223
4 1930s                               572
5 1940s                               683
```

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
- DESCENDANTS
- 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

```
CLOSINGPERIOD(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 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:

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

```
                            Patient Count
Q4 2003                                40
```

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

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

```
                            Patient Count
Q4 2010                                36
```

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:

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

```
                                    COUNT
Results                                 9
```

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

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

```
                          0 to 29          30 to 59               60+
1 None                       3,839             3,615               971
2 asthma                       308               282               113
3 CHD                            1                93               229
4 diabetes                      30               246               228
5 osteoporosis                   *                 *               200
```

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

```
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:

```
WITH SET myset AS 'diagd.MEMBERS'
SELECT COUNT(myset) ON 0 FROM patients WHERE aged.[0 to 29]
```

```
                                    COUNT
                                        5
```

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
```

```
                           Patient Count
Mar 24 1990                            1
```

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
```

```
                           Patient Count
Jan 24 1990                            *
```

See Also

- ANCESTOR
- CHILDREN
- DESCENDANTS
- 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.

  Note that no more than one of these sets can contain measures or expressions that evaluate to numbers. If both sets contain measures or expressions that evaluate to numbers, the DeepSee engine issues the error `Two measures cannot be crossjoined`.

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. Note that 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
```

```
                            Patient Count
 1 None->0 to 29                    3,839
 2 None->30 to 59                   3,615
 3 None->60+                          971
 4 asthma->0 to 29                    308
 5 asthma->30 to 59                   282
 6 asthma->60+                        113
 7 CHD->0 to 29                         1
 8 CHD->30 to 59                       93
 9 CHD->60+                           229
10 diabetes->0 to 29                   30
11 diabetes->30 to 59                 246
12 diabetes->60+                      228
13 osteoporosis->0 to 29                *
14 osteoporosis->30 to 59               *
15 osteoporosis->60+                  200
```

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
```

```
                            Patient Count
 1 None->0 to 29                    3,839
 2 None->30 to 59                   3,615
 3 None->60+                          971
 4 asthma->0 to 29                    308
 5 asthma->30 to 59                   282
 6 asthma->60+                        113
 7 CHD->0 to 29                         1
 8 CHD->30 to 59                       93
 9 CHD->60+                           229
10 diabetes->0 to 29                   30
11 diabetes->30 to 59                 246
12 diabetes->60+                      228
13 osteoporosis->60+                  200
```

For another example:

```
SELECT CROSSJOIN([GenD].[H1].[Gender].Members,
{[Measures].[%COUNT],[Measures].[Avg Age]}) ON 0,[DiagD].[H1].[Diagnoses].Members ON 1
FROM [Patients]
```

	Patient Coun	Avg Age	Patient Coun	Avg Age
1 None	419	35.61	412	32.36
2 asthma	51	37.12	33	27.79
3 CHD	17	70.47	19	62.42
4 diabetes	25	63.96	21	57.67
5 osteoporosis	20	80.55	2	74.50

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 ObjectScript.

Note: The CURRENTMEMBER function is not supported with the MEASURES dimension. That is, *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
```

```
                                Home ZIP
1 Cedar Falls                        iron
2 Centerville                video games
3 Cypress                         gravel
4 Elm Heights                    lettuce
5 Juniper                          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
```

```
                                   Export
1 Cedar Falls                        iron
2 Centerville                video games
3 Cypress                         gravel
4 Elm Heights                    lettuce
5 Juniper                          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:

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

	Export	Population
1 Cedar Falls	iron	90,000
2 Centerville	video games	49,000
3 Cypress	gravel	3,000
4 Elm Heights	lettuce	33,194
5 Juniper	wheat	10,333
6 Magnolia	bundt cake	4,503
7 Pine	spaghetti	15,060
8 Redwood	peaches	29,192
9 Spruce	mud	5,900

DESCENDANTS

Returns the members that are the descendants of a given member, within the specified level or levels.

Returned Type

This function returns a set of members.

Syntax and Details

```
DESCENDANTS(member_expression, level_expression, OPTIONAL_FLAG)
```

Or:

```
DESCENDANTS(member_expression, level_offset, OPTIONAL_FLAG)
```

- *member_expression* is an expression that evaluates to a member. In the following discussion, this member is the *target member*.
- *level_expression* is a level identifier. This must identify a level that is in the same hierarchy as the target member and that is lower in the hierarchy (more granular).

  As an alternative, you can specify *level_offset*, which is an integer that indicates a level relative to the one that contains the target member. For example, use 1 to specify the next lowest level.

  In the following discussion, the phrase *target level* refers to the level specified by *level_expression* or *level_offset*.
- *OPTIONAL_FLAG* specifies the level or levels whose members should be returned. This option describes a relationship to the target level. If specified, *OPTIONAL_FLAG* must be one of the following keywords, which are not case-sensitive:
  - `SELF` (the default) — Returns *only* descendents that are members of the target level.
  - `AFTER` — Returns descendents that are members of all levels *below* the target level.
  - `BEFORE` — Returns all descendents *up to and including* the target member. That is, it returns descendents that are members of all levels *above* the target level, and it also returns the target member.
  - `BEFORE_AND_AFTER` — Returns descendants from both *above* and *below* the target level, but does *not* include members from the target level.
  - `SELF_AND_AFTER` — Returns descendents that are members of the target level, as well as descendents from all levels *below* it.
  - `SELF_AND_BEFORE` — Returns descendents that are members of the target level, descendents that belong to all levels *above* the specified level, and the target member.
  - `SELF_BEFORE_AFTER` — Returns descendents from all levels, *including* the target member.

    In this case, the target level does not affect the outcome.

When the function returns members of more than one level, the hierarchy affects the order of the members as follows: A member of a higher level is followed by its children from the next lowest level, followed by the next member of the higher level, and so on. See the example for `SELF_AND_AFTER`.

Note: This implementation of DESCENDANTS does not support the optional LEAVES flag.

Example

For reference, in the Patients cube, the BirthD dimension contains the following levels, from highest to lowest:

- [BirthD].[H1].[Decade]
- [BirthD].[H1].[Year]
- [BirthD].[H1].[Quarter Year] (which represents year plus quarter)
- [BirthD].[H1].[Period] (which represents year plus month)
- [BirthD].[H1].[Date] (which represents year plus month plus day)

The following example gets all the descendents of the year 1990, within the [BirthD].[H1].[Period] level:

```
SELECT DESCENDANTS(birthd.1990,birthd.period) ON 1 FROM patients
```

```
 1 Jan-1990                              *
 2 Feb-1990                              2
 3 Mar-1990                              1
 4 Apr-1990                              1
 5 May-1990                              1
 6 Jun-1990                              *
 7 Jul-1990                              2
 8 Aug-1990                              2
 9 Sep-1990                              1
10 Oct-1990                              3
11 Nov-1990                              1
12 Dec-1990                              *
```

This example uses the default for *OPTIONAL_FLAG* (SELF), so the function returns only descendents of 1990 that are members of the period level.

The following variation uses NON EMPTY and thus filters out periods when no patients were born:

```
SELECT NON EMPTY DESCENDANTS(birthd.1990,birthd.period) ON 1 FROM patients
```

```
1 Feb-1990                               2
2 Mar-1990                               1
3 Apr-1990                               1
4 May-1990                               1
5 Jul-1990                               2
6 Aug-1990                               2
7 Sep-1990                               1
8 Oct-1990                               3
9 Nov-1990                               1
```

The period level is two levels below the year level, and the following query (which uses *level_offset* as 2) is equivalent to the first query:

```
SELECT DESCENDANTS(birthd.1990,2) ON 1 FROM patients
```

```
 1 Jan-1990                              *
 2 Feb-1990                              2
 3 Mar-1990                              1
 4 Apr-1990                              1
 5 May-1990                              1
 6 Jun-1990                              *
 7 Jul-1990                              2
 8 Aug-1990                              2
 9 Sep-1990                              1
10 Oct-1990                              3
11 Nov-1990                              1
12 Dec-1990                              *
```

The next variation uses AFTER:

```
SELECT DESCENDANTS(birthd.1990,birthd.period,AFTER) ON 1 FROM patients
```

```
  1 Jan 1 1990                           *
  2 Jan 2 1990                           *
  3 Jan 3 1990                           *
...
363 Dec 29 1990                          *
364 Dec 30 1990                          *
365 Dec 31 1990                          *
```

This example returns descendents of 1990 of all levels *below* the period level. In this case, there is only one lower level: date, which corresponds to year plus month plus day of the month.

The next variation uses SELF_AND_AFTER. This example returns members of more than one level and demonstrates the order in which these members are returned.

```
SELECT DESCENDANTS(birthd.1990,birthd.period,SELF_AND_AFTER) ON 1 FROM patients
```

```
  1 Jan-1990                              *
  2 Jan 1 1990                            *
  3 Jan 2 1990                            *
  4 Jan 3 1990                            *
...
 33 Feb-1990                              2
 34 Feb 1 1990                            *
 35 Feb 2 1990                            *
 36 Feb 3 1990                            1
...
346 Dec-1990                              *
347 Dec 1 1990                            *
348 Dec 2 1990                            *
349 Dec 3 1990                            *
...
377 Dec 31 1990                           *
```

The next variation uses BEFORE:

```
SELECT DESCENDANTS(birthd.1990,birthd.period,BEFORE) ON 1 FROM patients
```

```
1 1990                                   14
2 Q1 1990                                 3
3 Q2 1990                                 2
4 Q3 1990                                 5
5 Q4 1990                                 4
```

In this case, the query obtains all descendents of 1990 that are members of the levels *above* the period level (that is, it returns members of the quarter year level). Notice that 1990 is also returned.

See Also

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

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
```

```
                            Patient Count
 1 Juniper                          1,197
 2 Cedar Falls                      1,188
 3 Centerville                      1,155
 4 Cypress                          1,221
 5 Elm Heights                      1,266
 6 Juniper                          1,197
 7 Magnolia                         1,156
 8 Pine                             1,139
 9 Redwood                          1,144
10 Spruce                           1,135
```

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
```

```
                            Patient Count
1 Juniper                           1,197
2 Cedar Falls                       1,188
3 Centerville                       1,155
4 Cypress                           1,221
5 Elm Heights                       1,266
6 Magnolia                          1,156
7 Pine                              1,139
8 Redwood                           1,144
9 Spruce                            1,135
```

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
```

```
                              Patient Count
1 eggs                                 451
2 soy                                  462
```

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
```

```
                              Patient Count
1 eggs                                 451
2 eggs                                 451
3 soy                                  462
```

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
                              Patient Count
1 Cedar Falls                          1,188
2 Centerville                          1,155
3 Cypress                              1,221
4 Elm Heights                          1,266
5 Juniper                              1,197
6 Magnolia                             1,156
```

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

                              Patient Count
1 Cedar Falls                          1,188
2 Centerville                          1,155
3 Cypress                              1,221
4 Elm Heights                          1,266
5 Juniper                              1,197
6 Magnolia                             1,156
7 Pine                                 1,139
8 Redwood                              1,144
9 Spruce                               1,135
```

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

                              Patient Count
1 asthma                                 746
2 diabetes                               555
```

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

                                    ZIP
1 Cedar Falls                    90,000
2 Centerville                    49,000
3 Elm Heights                    33,194
4 Redwood                        29,192
```

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 an InterSystems 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
```

```
                              Patient Count
1 None                                8,425
2 asthma                                703
```

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:

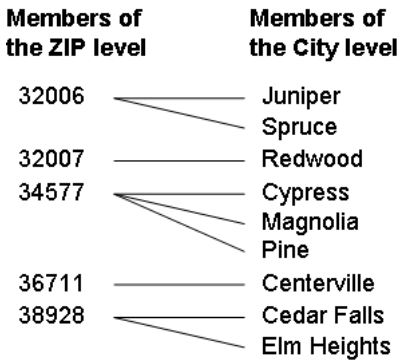
- *member_expression* is an expression that returns a member.

  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:



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
- DESCENDANTS
- 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

                           Patient Count
Jan-2003                              10
```

See Also

- CHILDREN
- COUSIN
- DESCENDANTS
- 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, optional_sample_flag)
```

- *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.
- *optional_sample_flag*, if included, is `SAMPLE`

  This optional flag has an effect only in the case where the *set_expression* contains multiple nested CROSSJOIN expressions. In such cases, by default, DeepSee does not attempt to determine the subset until *set_expression* has been fully evaluated and all the set elements are known. If you include the *optional_sample_flag*, DeepSee truncates the results as each CROSSJOIN is evaluated; this approach can be considerably faster but can result in a subset that contains fewer elements than given by the *optional_integer_expression*.

  Note that when you use drag and drop actions in the Analyzer to create queries that use the HEAD function for a set that contains multiple nested CROSSJOIN expressions, the Analyzer automatically adds this flag.

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
                              Patient Count
1 1910s                                  71
2 1920s                                 223
3 1930s                                 572
```

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*.

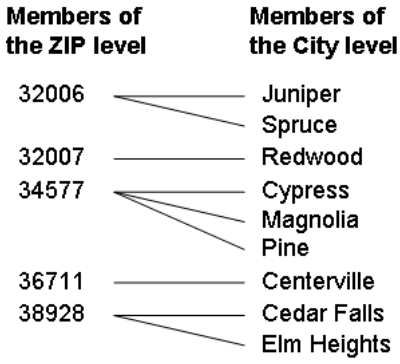
If the set members are in different hierarchies, the order of the hierarchies themselves is indeterminate. That is, if some members are from hierarchy A and the others are from hierarchy B, the A members will be listed consecutively in hierarchical order and the B members will be listed consecutively 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:



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:

```
SELECT MEASURES.[%COUNT] ON 0, HIERARCHIZE(homed.members) ON 1 FROM patients
                                Patient Count
 1 32006                                2,272
 2 Juniper                              1,155
 3 Spruce                               1,117
 4 32007                                1,111
 5 Redwood                              1,111
 6 34577                                3,399
 7 Cypress                              1,150
 8 Magnolia                             1,111
 9 Pine                                 1,138
10 36711                                1,069
11 Centerville                          1,069
12 38928                                2,149
13 Cedar Falls                          1,045
14 Elm Heights                          1,104
```

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
                                Patient Count
1 36711                                 1,069
2 Spruce                                1,117
3 38928                                 2,149
4 Elm Heights                           1,104
```

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
                                Patient Count
1 36711                                 1,069
2 Spruce                                1,117
3 Elm Heights                           1,104
4 38928                                 2,149
```

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:

```
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:

```
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:

```
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
```

The following example examines the name of the member and conditionally suppresses display of a measure:

```
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

                                      iif
1 Cedar Falls                       1,120
2 Centerville                       1,106
3 Cypress                           1,139
4 Elm Heights                       1,078
5 Juniper                           1,109
6 Magnolia                          1,122
7 Pine
8 Redwood                           1,128
9 Spruce                            1,108
```

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

```
INTERSECT(set_expression1, set_expression2, ALL)
```

Or:

```
INTERSECT(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 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:

```
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
1 soy                                    462
2 wheat                                  479
```

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

```
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
1 soy                                    462
2 wheat                                  479
3 wheat                                  479
```

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

```
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
1 Centerville                          1,155
2 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 an InterSystems 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
```

```
                                    ISNULL
 1 1910s                               None
 2 1920s                                  1
 3 1930s                                 12
 4 1940s                                  3
 5 1950s                                 10
 6 1960s                                  3
 7 1970s                               None
 8 1980s                                  1
 9 1990s                               None
10 2000s                               None
11 2010s                               None
```

See Also

- IIF
- %LABEL

LAG

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*.)

Within any time dimension, this function is more useful for a timeline-based time level (such as Period, which groups records by year and month) than for a date-part-based time level (such as Month, which groups records only by month). If the level is based on a date part, this function returns null when it refers to a level beyond the end of the set; see an example of a similar scenario in PREVMEMBER. For a fuller discussion, see "Introduction to Time Levels," in *Using MDX with DeepSee*.

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
```

```
                              Patient Count
1 1948                                   10
2 1949                                    4
3 1950                                   12
4 1951                                    8
5 1952                                    6
```

The following query uses LAG:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.LAG(1) ON 1 FROM patients
                              Patient Count
1950                                     12
```

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.LAG(2) ON 1 FROM patients
                              Patient Count
1949                                      4
```

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
```

```
                           Patient Count
Cypress                              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

                              Patient Count
Pine                                 1,100
```

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
                              Patient Count
Dec-1950                                 8
```

See Also

- CHILDREN
- COUSIN
- DESCENDANTS
- FIRSTSIBLING
- FIRSTCHILD
- LASTSIBLING
- PARENT
- SIBLINGS

LASTSIBLING

Returns the last sibling of the given member.

Returned Type

This function returns a member.

Syntax and Details

```
member_expression.LASTSIBLING
```

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 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
- DESCENDANTS
- 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*.)

Within any time dimension, this function is more useful for a timeline-based time level (such as Period, which groups records by year and month) than for a date-part-based time level (such as Month, which groups records only by month). If the level is based on a date part, this function returns null when it refers to a level beyond the end of the set; see an example of a similar scenario in NEXTMEMBER. For a fuller discussion, see "Introduction to Time Levels," in *Using MDX with DeepSee*.

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
```

```
                              Patient Count
1 1948                                   10
2 1949                                    4
3 1950                                   12
4 1951                                    8
5 1952                                    6
```

The following query uses LEAD:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.LEAD(1) ON 1 FROM patients
```

```
                              Patient Count
1949                                      4
```

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.LEAD(3) ON 1 FROM patients

                            Patient Count
1951                                    8
```

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:

```
SELECT MEASURES.[%COUNT] ON 0, homed.city.Magnolia.LEAD(1) ON 1 FROM patients

                            Patient Count
Pine                                  114
```

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.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

Looks up a given key in a term list and returns a substitute string. This function enables you to perform string replacements within a query. This function is an InterSystems extension to MDX.

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 *Advanced DeepSee Modeling 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:

Terms

key	value
Atlanta	Braves
Boston	Red Sox
New York	Yankees

Here is a simple query that uses this term list:

```
SELECT Lookup("Teams",Outlet.Boston.Properties("NAME")) ON ROWS FROM HOLEFOODS
Lookup                              Red Sox
```

Here is a more complex query:

```
SELECT Lookup("Teams",Outlet.CURRENTMEMBER.Properties("NAME"),"No Team") ON 0,
outlet.city.MEMBERS ON 1 FROM HOLEFOODS

                                    Lookup
 1 Amsterdam                        No Team
 2 Antwerp                          No Team
 3 Atlanta                           Braves
 4 Bangalore                        No Team
 5 Barcelona                        No Team
 6 Beijing                          No Team
 7 Berlin                           No Team
 8 Boston                           Red Sox
 9 Brasilia                         No Team
...
```

See Also

- %LOOKUP (which contains "Comparison with Other Term List Functions")
- %TERMLIST

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

```
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
                     Patient Count    Encounter Count     Avg Test Score
 1 1910s                        80              5,359              75.17
 2 1920s                       227             12,910              74.20
 3 1930s                       567             33,211              74.67
 4 1940s                       724             38,420              73.39
 5 1950s                     1,079             46,883              73.72
 6 1960s                     1,475             57,814              74.16
 7 1970s                     1,549             49,794              74.35
 8 1980s                     1,333             35,919              74.13
 9 1990s                     1,426             29,219              74.79
10 2000s                     1,406             20,072              74.95
11 2010s                       134              1,346              73.55
```

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
                     Patient Count    Encounter Count     Avg Test Score
MAX                         1,549             57,814              75.17
```

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.

Tip: To instead find the median value across the lowest-level records, use the %KPI function with the sample plugin class %DeepSee.PlugIn.Median. See the sample dashboards in the `KPIs & Plugins` folder in the SAMPLES namespace.

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
                     Patient Count    Encounter Count     Avg Test Score
 1 1910s                        80             5,359              75.17
 2 1920s                       227            12,910              74.20
 3 1930s                       567            33,211              74.67
 4 1940s                       724            38,420              73.39
 5 1950s                     1,079            46,883              73.72
 6 1960s                     1,475            57,814              74.16
 7 1970s                     1,549            49,794              74.35
 8 1980s                     1,333            35,919              74.13
 9 1990s                     1,426            29,219              74.79
10 2000s                     1,406            20,072              74.95
11 2010s                       134             1,346              73.55
```

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
                     Patient Count    Encounter Count     Avg Test Score
MEDIAN                       1,079            33,211              74.20
```

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
                                    MEDIAN
                                      1,079
```

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
                                Patient Count
1 32006                                 2,272
2 32007                                 1,111
3 34577                                 3,399
4 36711                                 1,069
5 38928                                 2,149
```

The following query displays all members of all levels in the `homed.h1` hierarchy as rows:

```
 1 32006                                2,272
 2 Juniper                              1,155
 3 Spruce                               1,117
 4 32007                                1,111
 5 Redwood                              1,111
 6 34577                                3,399
 7 Cypress                              1,150
 8 Magnolia                             1,111
 9 Pine                                 1,138
10 36711                                1,069
11 Centerville                          1,069
12 38928                                2,149
13 Cedar Falls                          1,045
14 Elm Heights                          1,104
```

See Also

- ALLMEMBERS

MIN

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

```
MIN(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 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
                     Patient Count    Encounter Count     Avg Test Score
 1 1910s                        80             5,359              75.17
 2 1920s                       227            12,910              74.20
 3 1930s                       567            33,211              74.67
 4 1940s                       724            38,420              73.39
 5 1950s                     1,079            46,883              73.72
 6 1960s                     1,475            57,814              74.16
 7 1970s                     1,549            49,794              74.35
 8 1980s                     1,333            35,919              74.13
 9 1990s                     1,426            29,219              74.79
10 2000s                     1,406            20,072              74.95
11 2010s                       134             1,346              73.55
```

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

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
MIN(birthd.decade.MEMBERS) ON 1 FROM patients
                     Patient Count    Encounter Count     Avg Test Score
MIN                            80             1,346              73.39
```

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
```

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).

Within any time dimension, this function is more useful for a timeline-based time level (such as Period, which groups records by year and month) than for a date-part-based time level (such as Month, which groups records only by month). See the examples. For a fuller discussion, see "Introduction to Time Levels," in *Using MDX with DeepSee*.

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
```

```
                              Patient Count
1 1948                                   10
2 1949                                    4
3 1950                                   12
4 1951                                    8
5 1952                                    6
```

The following query uses NEXTMEMBER:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.NEXTMEMBER ON 1 FROM patients
```

```
                              Patient Count
1949                                      4
```

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1949.NEXTMEMBER ON 1 FROM patients
```

```
                              Patient Count
1950                                     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
```

```
                            Patient Count
Pine                                  114
```

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
```

```
                            Patient Count
                                        *
```

For time level that is based on *part* of the date, this function returns null for the last member of the level. Consider the Month level, which groups records only by month. When we use this function with December, the engine returns null:

```
SELECT [BirthQD].[H1].[Month].[December].NEXTMEMBER ON 1 FROM patients
```

```
                                        *
```

For a fuller discussion, see "Introduction to Time Levels," in *Using MDX with DeepSee*.

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

```
NONEMPTYCROSSJOIN(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. 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.

Example

For example:

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

```
                            Patient Count
 1 None->0 to 29                      363
 2 None->30 to 59                     348
 3 None->60+                          120
 4 asthma->0 to 29                     36
 5 asthma->30 to 59                    37
 6 asthma->60+                         11
 7 CHD->30 to 59                       13
 8 CHD->60+                            23
 9 diabetes->0 to 29                    1
10 diabetes->30 to 59                  20
11 diabetes->60+                       25
12 osteoporosis->60+                   22
```

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
```

```
                            Patient Count
Q1 2003                                35
```

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
```

```
                            Patient Count
Q1 2000                                33
```

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
                            Avg Test Score
1 Pine                                75.67
2 Magnolia                            74.65
3 Cypress                             74.61
4 Centerville                         74.85
5 Cedar Falls                         74.62
6 Elm Heights                         74.36
7 Juniper                             74.52
8 Spruce                              74.14
9 Redwood                             74.16
```

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
                            Avg Test Score
1 Pine                                75.67
2 Centerville                         74.85
3 Magnolia                            74.65
4 Cedar Falls                         74.62
5 Cypress                             74.61
6 Juniper                             74.52
7 Elm Heights                         74.36
8 Redwood                             74.16
9 Spruce                              74.14
```

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
                        Avg Test Score           stringtest
1 Pine                           75.67                 high
2 Cedar Falls                    74.62                  low
3 Centerville                    74.85                  low
4 Cypress                        74.61                  low
5 Elm Heights                    74.36                  low
6 Juniper                        74.52                  low
7 Magnolia                       74.65                  low
8 Redwood                        74.16                  low
9 Spruce                         74.14                  low
```

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
38928                               2,276
```

To see a picture of the hierarchy used in this example, see the FIRSTCHILD function.

See Also

- ANCESTOR
- CHILDREN
- COUSIN
- DESCENDANTS
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- LASTSIBLING
- SIBLINGS

PERCENTILE

Evaluates a given expression (or the current measure), across all elements of a set, and returns the value that is at a given percentile level.

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 the value that is at the given percentile.

Tip: To instead find a value at the percentile across the lowest-level records, use the %KPI function with the sample plugin class %DeepSee.PlugIn.Percentile. See the sample dashboards in the `KPIs & Plugins` folder in the SAMPLES namespace.

Example

For reference, 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:

```
SELECT MEASURES.[%COUNT] ON 0, ORDER(birthd.year.MEMBERS,MEASURES.[%COUNT],BASC) ON 1 FROM patients
```

```
                            Patient Count
 1 1916                                 1
 2 1921                                 1
 3 1922                                 1
 4 1925                                 1
 5 1941                                 1
 6 1914                                 2
...
82 1967                                18
83 1969                                18
84 1973                                18
85 1978                                18
86 1979                                18
87 1981                                18
88 2002                                18
89 2009                                18
90 1968                                19
91 1998                                21
92 1991                                23
93 2003                                23
94 1977                                25
```

Next, the following query shows the 5th percentile value for these members:

```
SELECT MEASURES.[%COUNT] ON 0, PERCENTILE(birthd.year.MEMBERS,,5) ON 1 FROM patients
```

```
                            Patient Count
5 Percentile                            1
```

That is, the 5th percentile consists of birth years that have at most 1 patient.

The following query shows the 95th percentile instead:

```
SELECT MEASURES.[%COUNT] ON 0, PERCENTILE(birthd.year.MEMBERS,,95) ON 1 FROM patients
```

```
                            Patient Count
95 Percentile                          18
```

That is, the 95th percentile consists of birth years that have 18 or fewer patients.

For another example, we use the second argument for PERCENTILE:

```
SELECT PERCENTILE(birthd.year.MEMBERS,MEASURES.[%COUNT],50) ON 1 FROM patients
```

```
50 Percentile                          10
```

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.

Example

For reference, 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:

```
SELECT MEASURES.[%COUNT] ON 0, ORDER(birthd.year.MEMBERS,MEASURES.[%COUNT],BASC) ON 1 FROM patients
```

```
                            Patient Count
 1 1916                                 1
 2 1921                                 1
 3 1922                                 1
 4 1925                                 1
 5 1941                                 1
 6 1914                                 2
 7 1918                                 2
 8 1920                                 3
 9 1927                                 3
10 1931                                 3
11 1934                                 3
12 1926                                 4
13 1932                                 4
14 1949                                 4
15 1955                                 4
16 1956                                 4
17 1928                                 5
18 1930                                 5
19 1937                                 5
20 1938                                 5
21 1966                                 5
22 1929                                 6
23 1940                                 6
24 1952                                 6
25 1939                                 7
...
93 2003                                23
94 1977                                25
```

Next, the following query shows the percentile rank of a member that has 5 patients:

```
SELECT PERCENTILERANK(birthd.year.MEMBERS,MEASURES.[%Count],5) ON 1 FROM patients

Rank of 5                              19.68
```

That is, if you consider birth years that have 5 patients or fewer, these birth years constitute 19.68% of the set of years.

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
```

```
                              Patient Count
1 Q1 2003                                35
2 Q2 2003                                44
3 Q3 2003                                43
```

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
```

```
                              Patient Count
 1 Q1 2000                               33
 2 Q2 2000                               33
 3 Q3 2000                               45
 4 Q4 2000                               39
 5 Q1 2001                               37
 6 Q2 2001                               40
 7 Q3 2001                               36
 8 Q4 2001                               37
 9 Q1 2002                               39
10 Q2 2002                               35
11 Q3 2002                               40
12 Q4 2002                               38
13 Q1 2003                               35
14 Q2 2003                               44
15 Q3 2003                               43
```

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

```
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:

```
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:

```
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:

```
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).

Within any time dimension, this function is more useful for a timeline-based time level (such as Period, which groups records by year and month) than for a date-part-based time level (such as Month, which groups records only by month). See the examples. For a fuller discussion, see "Introduction to Time Levels," in *Using MDX with DeepSee*.

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
```

```
                              Patient Count
1 1948                                   10
2 1949                                    4
3 1950                                   12
4 1951                                    8
5 1952                                    6
```

The following query uses PREVMEMBER:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.PREVMEMBER ON 1 FROM patients
```

```
                              Patient Count
1950                                     12
```

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1950.PREVMEMBER ON 1 FROM patients
```

```
                              Patient Count
1949                                      4
```

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
```

```
                            Patient Count
Cypress                                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.PREVMEMBER ON 1 FROM patients
```

```
                            Patient Count
                                        *
```

For time level that is based on *part* of the date, this function returns null for the first member of the level. Consider the Month level, which groups records only by month. When we use this function with January, the engine returns null:

```
SELECT [BirthQD].[H1].[Month].[January].PREVMEMBER ON 1 FROM patients
```

```
                                        *
```

For an example that uses PREVMEMBER to get the number of units sold in a previous time period, see %CELLZERO.

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

```
member_expression.PROPERTIES(property_name,default_value))
```

Where:

- *member_expression* is an expression that returns a member.

  This expression cannot refer to a measure.

- *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.

- *default_value* is an optional value to return if the member does not have a value for this property. If you omit this argument and if the given member does not have the property, the function returns `@NOPROPERTY`.

  This argument is an InterSystems extension to MDX.

Names of properties are not case-sensitive.

Example

The following example gets the value of the KEY property, which is an intrinsic property:

```
SELECT docd.h1.CURRENTMEMBER.PROPERTIES("KEY") ON 0, docd.[doctor].MEMBERS ON 1 FROM patients
```

```
                                  Doctor
  1 None                          <null>
  2 Adam, Dan                         41
  3 Adam, Danielle                   391
  ...
```

The following variation uses %LABEL to provide a better caption:

```
SELECT %LABEL(docd.h1.CURRENTMEMBER.PROPERTIES("key"),"key") ON 0,
docd.doctor.MEMBERS ON 1 FROM patients
```

```
                                   key
  1 None                          <null>
  2 Adam, Dan                         41
  3 Adam, Danielle                   391
  ...
```

The following example uses CURRENTMEMBER and iterates through the ZIP codes to retrieve the values of two intrinsic properties: ID and LEVEL_NUMBER:

```
WITH SET test AS '{homed.h1.CURRENTMEMBER.PROPERTIES("id"),
homed.h1.CURRENTMEMBER.PROPERTIES("level_number")}'
SELECT test ON 0, homed.zip.MEMBERS ON 1 FROM patients
```

```
                            Home ZIP            Home ZIP
1 32006                            2                   1
2 32007                            4                   1
3 34577                            1                   1
4 36711                            5                   1
5 38928                            3                   1
```

As a variation, the following query uses %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
```

```
                                  id         level_number
1 32006                             2                    1
2 32007                             4                    1
3 34577                             1                    1
4 36711                             5                    1
5 38928                             3                    1
```

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
Results                                 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 an InterSystems extension to MDX.

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
```

```
                                    ROUND
1 Group A                             0.97
2 Group B                             1.06
3 None                                0.97
```

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
```

```
                              Patient Count
1 Cypress                             1,089
2 Magnolia                            1,073
3 Pine                                1,039
```

To see a picture of the hierarchy used in this example, see the FIRSTCHILD function.

See Also

- CHILDREN
- COUSIN
- DESCENDANTS
- 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
                       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 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
                       Patient Count      Encounter Count
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:

```
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:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
STDDEVP(birthd.decade.MEMBERS) ON 1 FROM patients
                      Patient Count      Encounter Count
STDDEVP                      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:

```
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 STDDEVP.

See Also

- STDDEVP

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

```
SUBSET(set_expression, first_element_position, optional_element_count)
```

Where:

- *set_expression* is an expression that evaluates to a set.
- *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.
- *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 *first_element_position* and all elements that follow it.

Example

```
SELECT MEASURES.[%COUNT] ON 0, SUBSET(homed.city.MEMBERS, 0, 3) ON 1 FROM patients
```

```
                              Patient Count
1 Cedar Falls                         1,188
2 Centerville                         1,155
3 Cypress                             1,221
```

In contrast, consider this example, which shows the complete set:

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

```
                              Patient Count
1 Cedar Falls                         1,188
2 Centerville                         1,155
3 Cypress                             1,221
4 Elm Heights                         1,266
5 Juniper                             1,197
6 Magnolia                            1,156
7 Pine                                1,139
8 Redwood                             1,144
9 Spruce                              1,135
```

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
                     Patient Count    Encounter Count     Avg Test Score
 1 1910s                        80              5,359              75.17
 2 1920s                       227             12,910              74.20
 3 1930s                       567             33,211              74.67
 4 1940s                       724             38,420              73.39
 5 1950s                     1,079             46,883              73.72
 6 1960s                     1,475             57,814              74.16
 7 1970s                     1,549             49,794              74.35
 8 1980s                     1,333             35,919              74.13
 9 1990s                     1,426             29,219              74.79
10 2000s                     1,406             20,072              74.95
11 2010s                       134              1,346              73.55
```

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
                     Patient Count    Encounter Count     Avg Test Score
SUM                         10,000            330,947             817.08
```

Here, each value is the sum of the values in a column in the preceding query. For example, the `Patient Count` value is the sum 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
                              Patient Count
1 1990s                               1,413
2 2000s                               1,433
3 2010s                                 155
```

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

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

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
                              Patient Count
1 1970s                              1,520
2 1960s                              1,500
3 2000s                              1,433
```

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
```

```
                            Patient Count
 1 2000s                               157
 2 1980s                               155
 3 1990s                               144
 4 1960s                               136
 5 1970s                               128
 6 1950s                               107
 7 1930s                                56
 8 1940s                                54
 9 2010s                                44
10 1920s                                13
11 1910s                                 6
```

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
```

```
                            Patient Count
1 2000s                               157
2 1980s                               155
3 1990s                               144
4 1960s                               136
```

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

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
                              Patient Count
1970s                                 1,520
```

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

```
UNION(set_expression1, set_expression2, ALL)
```

Or:

```
UNION(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 returned set includes any duplicate elements, only the first of those is included.

Example

Consider the following query which defines two named sets:

```
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
                              Patient Count
1 eggs                                  451
2 soy                                   462
3 wheat                                 479
4 dairy products                        463
5 pollen                                447
```

This query shows all the members that are in `set1` and `set2`.

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

```
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
                              Patient Count
1 eggs                                  451
2 soy                                   462
3 wheat                                 479
4 dairy products                        463
5 pollen                                447
6 soy                                   462
7 wheat                                 479
```

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

```
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
          Patient Count
1 Juniper                           1,197
2 Spruce                            1,135
3 Centerville                       1,155
4 Redwood                           1,144
5 Magnolia                          1,156
6 Cedar Falls                       1,188
7 Elm Heights                       1,266
8 Pine                              1,139
```

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

```
VAR(set_expression, optional_numeric_expression)
```

Or:

```
VAR(set_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 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
                       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
                       Patient Count      Encounter Count
VAR                        335,710.89       338,609,051.69
```

Here, each value is the variance of the values in a column in the preceding query. For example, the `Patient Count` value is the variance of the `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
                       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 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
                       Patient Count      Encounter Count
VARP                       305,191.72       307,826,410.63
```

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)
```

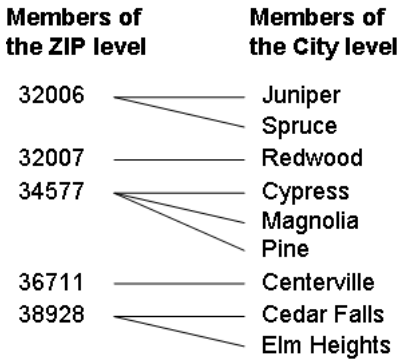
Where:

- *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, for reference, the following figure shows the hierarchy used in this example:



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
                              Patient Count
1 32006                                2,272
2 Spruce                               1,117
3 34577                                3,399
4 Cypress                              1,150
5 Pine                                 1,138
```

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
                              Patient Count
1 32006                               1,117
2 Spruce                              1,117
3 34577                               2,288
4 Cypress                             1,150
5 Pine                                1,138
```

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
                              Patient Count
1 32006 (included cities)             1,117
2 Spruce                              1,117
3 34577 (included cities)             2,288
4 Cypress                             1,150
5 Pine                                1,138
```

The values shown are the same as in the preceding query, but each ZIP code is shown with the trailing string `(included cities)`.

See Also

- HIERARCHIZE

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.

Property	Details	Example
KEY	Key for the member, for use with the syntax &[key] when referring to the member, as discussed in "Member Expressions." For details, see "Key Values."	`Juniper`
ID	Internal ID of the member.	`6`
NAME	Display name of the member. This is taken from the source value or (if present) from the value of a property marked `isName="true"`	`Juniper`
MEMBER_NAME		
CAPTION		
CUBE_NAME	Uppercase version of the logical name of the cube to which this member belongs. This is controlled by the `name` attribute of the cube.	`PATIENTS`
LEVEL_NUMBER	Number of the level to which this member belongs. Level 0 is the All level.	`3`
LEVEL_CAPTION	Localized version of the `displayName` of the current level. The `displayName` attribute is specified within the cube definition.	`City`
LEVEL	Uppercase version of the logical name of the level. This is controlled by the `name` attribute of the level in the cube definition.	`CITY`
HIERARCHY_CAPTION	Localized version of the `displayName` of the current hierarchy. The `displayName` attribute is specified within the cube definition.	`H1`
HIERARCHY	Uppercase version of the logical name of the hierarchy. This is controlled by the `name` attribute of the hierarchy in the cube definition.	`H1`
DIMENSION_CAPTION	Localized version of the `displayName` of the current dimension. The `displayName` attribute is specified within the cube definition.	`HomeD`
DIMENSION	Uppercase version of the logical name of the dimension. This is controlled by the `name` attribute of the dimension in the cube definition.	`HOMED`

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.

Scenario	Format of KEY value	Example Member Name*	Corresponding KEY Value
Null member	`<null>`	`None`	`<null>`
All member	`*`	`All Patient Addresses`	`*`
Member of a level that uses the HourNumber time function	Integer that represents the hour number	`2`	`2`
Member of a level that uses the DayMonthYear time function	Integer that represents the date in **$HOROLOG** format	`Dec 27 2004`	`59896`
Member of a level that uses the DayNumber time function	Integer that represents the day number	`21`	`21`
Member of a level that uses the WeekNumber time function	Integer that represents the day number	`6`	`6`
Member of a level that uses the MonthNumber time function	Integer that represents the month number	`July`	`7`
Member of a level that uses the WeekYear time function	String in the form YYYYMWW	`2012W06`	`2012W06`
Member of a level that uses the MonthYear time function	Integer in the form YYYYMM	`July 2010`	`201007`
Member of a level that uses the QuarterNumber time function	Integer that represents the quarter number	`Q3`	`3`
Member of a level that uses the QuarterYear time function	Integer in the form YYYYQQ	`Q4 2010`	`201004`
Member of a level that uses the Year time function	Four-digit integer that represents the year number	`2009`	`2009`
Member of a level that uses the Decade time function	Four-digit integer that represents the year number, followed by `s`	`1990s`	`1990s`
Member of a computed dimension	Member name, followed by a colon, followed by the SQL SELECT statement that retrieves the values	`member 1`	shown after this table***
Other scenarios	Source value of the member (which is also used as the member name by default)**	`Juniper`	`Juniper`
		`Sorenson,Violet`	`169`

*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*.

**In a well-defined cube, each member key is unique. See "Defining Member Keys and Names Appropriately" in *Defining DeepSee Models*.

***The key for a member of a computed dimension might be as follows:

```
&[member 1:select ID from DeepSee_Model_PatientsCube.Fact WHERE MxAge<50 AND DxHomeCity->PxName='Elm
Heights']
```

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 an InterSystems 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):

Expression	Returned member
`birthd.decade.NOW`	`birthd.decade.2010s`
`birthd.[quarter year].NOW`	`birthd.[quarter year].[Q2 2012]`
`birthqd.[quarter].NOW`	`birthd.[quarter year].Q2`
`birthtd.NOW`	`birthtd.4pm`

`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):

Expression	Returned member
`birthd.decade.[NOW-4]`	`birthd.decade.1970s`
`birthd.[quarter year].[NOW-4]`	`birthd.[quarter year].[Q2 2011]`
`birthqd.quarter.[NOW-4]`	`birthqd.quarter.Q2`

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:

Time Function...	Typical Members	Notes
QuarterNumber	Q4	These levels are independent of the year
MonthNumber	November	
DayNumber	24	This level is independent of the year and the part of the year
HourNumber	1am	These levels are independent of the day
MinuteNumber	01:24	

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 *nn*y*nn*m*nn*d, as follows:

- *nn* is a one or two digit integer.
- The optional unit *nn*y specifies the number of years.
- The optional unit *nn*m specifies the number of months.
- The optional unit *nn*d specifies the number of days.

The following table shows examples (created on 24 May 2012):

Expression	Returned member
`birthd.date.[NOW-4y3m2d]`	`birthd.date.[Feb 22 2008]`
`birthd.date.[NOW-1m]`	`birthd.date.[Apr 24 2012]`

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:

Expression	Returned member
`dateofsale.actual.daysold.[NOW-1y-1d]`	`dateofsale.actual.daysold.[May 25 2011]`

DeepSee automatically accounts for leap years. The internal logic also accounts for the varying lengths of the months.

A

Quick Function Reference

The following table summarizes the syntax and return type of each supported MDX function.

Function	Syntax	Return Type
%ALL	*member_expression*.%ALL	member
%CELL	%CELL(*relative_column_position*, *relative_row_position*)	number or string
%CELLZERO	%CELLZERO(*relative_column_position*, *relative_row_position*)	number or string
%FIRST	%FIRST(*set_expr*, *optional_numeric_expr*)	number
%KPI	%KPI(*kpi_name*, *kpi_prop_name*, *kpi_series_name*, *parm*, *value*, *parm*, *value*,...)	number
%LABEL	%LABEL(*MDX_expr*, *label*, *format_details*, *solve_order*, *cell_style*, *heading_style*)	same as *MDX_expr*
%LAST	%LAST(*set_expr*, *optional_numeric_expr*)	number
%LIST	%LIST(*set_expr*)	string (comma-separated list)
%LOOKUP	%LOOKUP(*termlist*, *key*, *field*, *default*)	number or string
%MDX	%MDX("*MDX select query*", *parm*, *value*, *parm*, *value*, *parm*, *value*,...)	number or string
%NOT	*member_expression*.%NOT	member
%OR	%OR(*set_expr*)	member
%SEARCH	%SEARCH.&[*comparison_expression*]	measure search expression
%SPACE	%SPACE()	empty space
%TERMLIST	%TERMLIST(*term_list_name*, `INCLUDE` | `EXCLUDE`)	set
%TIMERANGE	%TIMERANGE(*start_member*, *end_member*, `INCLUSIVE` | `EXCLUSIVE`)	member

Function	Syntax	Return Type
%TIMEWINDOW	%TIMEWINDOW(*set_expr*, *start_member*, *optional_end_member*)	set of members
%TOPMEMBERS	*level_expr*.%TOPMEMBERS *hierarchy_expr*.%TOPMEMBERS *dimension_expr*.%TOPMEMBERS	set of members
AGGREGATE	AGGREGATE(*set_expr*, *optional_numeric_expr*)	number
ALLMEMBERS	*level_expr*.ALLMEMBERS *hierarchy_expr*.ALLMEMBERS *dimension_expr*.ALLMEMBERS	set of members
ANCESTOR	ANCESTOR(*member_expr*, *ancestor_level*)	member
AVG	AVG(*set_expr*, *optional_numeric_expr*)	number
BOTTOMCOUNT	BOTTOMCOUNT(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
BOTTOMPERCENT	BOTTOMPERCENT(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
BOTTOMSUM	BOTTOMSUM(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
CHILDREN	*member_expr*.CHILDREN	set of members
CLOSINGPERIOD	CLOSINGPERIOD(*ancestor_level*, *member_expr*)	member
COUNT	COUNT(*set_expr*) COUNT(*set_expr*, EXCLUDEEMPTY)	number
COUSIN	COUSIN(*member_expr*, *higher_member_expr*)	member
CROSSJOIN	CROSSJOIN(*set_expr1*, *set_expr2*) NON EMPTY CROSSJOIN(*set_expr1*, *set_expr2*)	set of tuples
CURRENTMEMBER	*hierarchy_expr*.CURRENTMEMBER *dimension_expr*.CURRENTMEMBER	member
DESCENDANTS	DESCENDANTS(*member_expression*, *level_expression*, *OPTIONAL_FLAG*) DESCENDANTS(*member_expression*, *level_offset*, *OPTIONAL_FLAG*)	set of members
DISTINCT	DISTINCT(*set_expr*)	set
EXCEPT	EXCEPT(*set_expr1*, *set_expr2*, ALL) EXCEPT(*set_expr1*,*set_expr2*)	set
FILTER	FILTER(*set_expr*, *logical_expr*)	set

Function	Syntax	Return Type
FIRSTCHILD	*member_expr*.FIRSTCHILD	member
FIRSTSIBLING	*member_expr*.FIRSTSIBLING	member
HEAD	HEAD(*set_expr*, *optional_integer_expr*, *optional_sample_flag*)	set
HIERARCHIZE, HIERARCHISE	HIERARCHIZE(*set_expr*) HIERARCHIZE(*set_expr*, POST)	set of members
IIF	IIF(*logical_expr*, *expression1*, *expression2*)	number or string
INTERSECT	INTERSECT(*set_expr1*, *set_expr2*)	set
ISNULL	ISNULL(*scalar_expression*,*scalar_value_if_null*)	number or string
LAG	*member_expr*.LAG(*optional_nonnegative_integer_expr*)	member
LASTCHILD	*member_expr*.LASTCHILD	member
LASTSIBLING	*member_expr*.LASTSIBLING	member
LEAD	*member_expr*.LEAD(*optional_nonnegative_integer_expr*)	member
LOG	LOG(*numeric_expr*)	number
LOOKUP	LOOKUP(*term_list_name*, *lookup_value*, *default*, *alternative_field*)	string
MAX	MAX(*set_expr*, *optional_numeric_expr*)	number
MEDIAN	MEDIAN(*set_expr*, *optional_numeric_expr*)	number
MEMBERS	*level_expr*.MEMBERS *hierarchy_expr*.MEMBERS *dimension_expr*.MEMBERS	set of members
MIN	MIN(*set_expr*, *optional_numeric_expr*)	number
NEXTMEMBER	*member_expr*.NEXTMEMBER	member
NONEMPTYCROSSJOIN	NONEMPTYCROSSJOIN(*set_expr1*, *set_expr2*)	set of tuples
OPENINGPERIOD	OPENINGPERIOD(*ancestor_level*, *member_expr*)	member
ORDER	ORDER(*set_expr*, *ordering_expr*, ASC | DESC | BASC | BDESC) ORDER(*set_expr*, *ordering_expr*)	set
PARALLELPERIOD	PARALLELPERIOD(*level_expr*, *offset*, *member_expr*)	member
PARENT	*member_expr*.PARENT	member
PERCENTILE	PERCENTILE(*set_expr*, *numeric_expr*, *numeric_expr*, *optional_percentile_value*)	number
PERCENTILERANK	PERCENTILERANK(*set_expr*, *numeric_expr*, *comparison_value*)	number

Function	Syntax	Return Type
PERIODSTODATE	PERIODSTODATE(*ancestor_level*, *member_expr*)	set of members
POWER	POWER(*numeric_expr*,*numeric_expr_for_power*)	number
PREVMEMBER	*member_expr*.PREVMEMBER	member
PROPERTIES	*member_expr*.PROPERTIES(*property_name*)	string
RANK	RANK(*tuple_expr*, *set_expr*, *optional_numeric_expr*)	number
ROUND	ROUND(*numeric_expr*,*decimal_places*)	number
SIBLINGS	*member_expr*.SIBLINGS	set of members
SQRT	SQRT(*numeric_expr*)	number
STDDEV, STDEV	STDDEV(*set_expr*, *optional_numeric_expr*)	number
STDDEVP, STDEVP	STDDEVP(*set_expr*, *optional_numeric_expr*)	number
SUBSET	SUBSET(*set_expr*, *first_element_expr*, *optional_element_count*)	set
SUM	SUM(*set_expr*, *optional_numeric_expr*)	number
TAIL	TAIL(*set_expr*, *optional_integer_expr*)	set
TOPCOUNT	TOPCOUNT(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
TOPPERCENT	TOPPERCENT(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
TOPSUM	TOPSUM(*set_expr*, *element_count*, *optional_ordering_expr*)	set of members or tuples
UNION	UNION(*set_expr1*,*set_expr2*) UNION(*set_expr1*,*set_expr2*, ALL)	set
VAR, VARIANCE	VAR(*set_expr*, *optional_numeric_expr*)	number
VARP, VARIANCEP	VARP(*set_expr*, *optional_numeric_expr*)	number
VISUALTOTALS	VISUALTOTALS(*set_expr*, *optional_parent_name_pattern*)	set of members