Data Mining Extensions (DMX) Reference

SQL Server 2012 Books Online

Reference

Microsoft

Data Mining Extensions (DMX) Reference

SQL Server 2012 Books Online

Summary: Data Mining Extensions (DMX) is a language that you can use to create and work with data mining models in Microsoft SQL Server Analysis Services. You can use DMX to create the structure of new data mining models, to train these models, and to browse, manage, and predict against them. DMX is composed of data definition language (DDL) statements, data manipulation language (DML) statements, and functions and operators.

Category: Reference
Applies to: SQL Server 2012
Source: SQL Server Books Online (link to source content)
E-book publication date: June 2012

Contents

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) is a language that you can use to create and work with data mining models in Microsoft SQL Server Analysis Services. You can use DMX to create the structure of new data mining models, to train these models, and to browse, manage, and predict against them. DMX is composed of data definition language (DDL) statements, data manipulation language (DML) statements, and functions and operators.

Microsoft OLE DB for Data Mining Specification

The data mining features in Analysis Services are built to comply with the Microsoft OLE DB for Data Mining specification.

The Microsoft OLE DB for Data Mining specification defines the following:

- A structure to hold the information that defines a data mining model.
- A language for creating and working with data mining models.

The specification defines the basis of data mining as the data mining model virtual object. The data mining model object encapsulates all that is known about a particular mining model. The data mining model object is structured like an SQL table, with columns, data types, and meta information that describe the model. This structure lets you use the DMX language, which is an extension of SQL, to create and work with models.

For More Information: Understanding the Select Statement (DMX)

DMX Statements

You can use DMX statements to create, process, delete, copy, browse, and predict against data mining models. There are two types of statements in DMX: data definition statements and data manipulation statements. You can use each type of statement to perform different kinds of tasks.

The following sections provide more information about working with DMX statements:

- Data Definition Statements
- Data Manipulation Statements
- Query Fundamentals

Data Definition Statements

Use data definition statements in DMX to create and define new mining structures and models, to import and export mining models and mining structures, and to drop existing models from a database. Data definition statements in DMX are part of the data definition language (DDL).

You can perform the following tasks with the data definition statements in DMX:

- Create a mining structure by using the CREATE MINING STRUCTURE statement, and add a mining model to the mining structure by using the ALTER MINING STRUCTURE statement.
- Create a mining model and associated mining structure simultaneously by using the CREATE MINING MODEL statement to build an empty data mining model object.
- Export a mining model and associated mining structure to a file by using the EXPORT statement. Import a mining model and associated mining structure from a file that is created by the **EXPORT** statement by using the IMPORT statement.
- Copy the structure of an existing mining model into a new model, and train it with the same data, by using the SELECT INTO statement.
- Completely remove a mining model from a database by using the DROP MINING MODEL statement. Completely remove a mining structure and all its associated mining models from the database by using the DROP MINING STRUCTURE statement.

To learn more about the data mining tasks that you can perform by using DMX statements, see DMX Statement Reference.

Back to DMX Statements

Data Manipulation Statements

Use data manipulation statements in DMX to work with existing mining models, to browse the models and to create predictions against them. Data manipulation statements in DMX are part of the data manipulation language (DML).

You can perform the following tasks with the data manipulation statements in DMX:

- Train a mining model by using the INSERT INTO statement. This does not insert the actual source data into a data mining model object, but instead creates an abstraction that describes the mining model that the algorithm creates. The source query for an **INSERT INTO** statement is described in <source data query>.
- Extend the **SELECT** statement to browse the information that is calculated during model training and stored in the data mining model, such as statistics of the source data. Following are the clauses that you can include to extend the power of the **SELECT** statement:
  - SELECT DISTINCT FROM <model>
  - SELECT FROM <model>.CONTENT
  - SELECT FROM <model>.CASES
  - SELECT FROM <model>.SAMPLE_CASES
  - SELECT FROM <model>.DIMENSION_CONTENT
- Create predictions that are based on an existing mining model by using the PREDICTION JOIN clause of the **SELECT** statement. The source query for a **PREDICTION JOIN** statement is described in <source data query>.

- Remove all the trained data from a model or a structure by using the DELETE statement.

To learn more about the data mining tasks that you can perform by using DMX statements, see DMX Statement Reference.

Back to DMX Statements

DMX Query Fundamentals

The **SELECT** statement is the basis for most DMX queries. Depending on the clauses that you use with such statements, you can browse, copy, or predict against mining models. The prediction query uses a form of **SELECT** to create predictions based on existing mining models. Functions extend your ability to browse and query the mining models beyond the intrinsic capabilities of the data mining model.

You can use DMX functions to obtain information that is discovered during the training of your models, and to calculate new information. You can use these functions for many purposes, including to return statistics that describe the underlying data or the accuracy of a prediction, or to return an expanded explanation of a prediction.

For More Information: Understanding the Select Statement (DMX), Mapping Functions to Query Types (DMX), Prediction Queries, DMX Function Reference

Back to DMX Statements

See Also

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Structure and Usage of DMX Prediction Queries

In Microsoft SQL Server Analysis Services, you can use the prediction query in Data Mining Extensions (DMX) to predict unknown column values in a new dataset, based on the results of a mining model.

The type of query you use depends on what information you want to obtain from a model. If you want to create simple predictions in real time, for example to know if a potential customer on a Web site fits the persona of a bike buyer, you would use a singleton query. If you want to create a batch of predictions from a set of cases that are contained within a data source, you would use a regular prediction query.

Prediction Types

You can use DMX to create the following types of predictions:

Prediction join

Use to create predictions on input data based on the patterns that exist in the mining model. This query statement must be followed by an **ON** clause that supplies the join conditions between the mining model columns and the input columns.

Natural prediction join

Use to create predictions that are based on column names in the mining model that exactly match the column names in the table on which you are performing the query. This query statement does not require an **ON** clause, because the join condition is automatically generated based on the matching names between the mining model columns and the input columns.

Empty prediction join

Use to discover the most likely prediction, without having to supply input data. This returns a prediction that is based only on the content of the mining model.

Singleton query

Use to create a prediction by feeding the data to the query. This statement is useful because you can feed a single case to the query, to get a result back quickly. For example, you can use the query to predict whether someone who is female, age 35, and married would be likely to purchase a bicycle. This query does not require an external data source.

Query Structure

To build a prediction query in DMX, you use a combination of the following elements:

- **SELECT [FLATTENED]**
- **TOP**
- **FROM** <*model*> **PREDICTION JOIN**
- **ON**
- **WHERE**
- **ORDER BY**

The **SELECT** element of a prediction query defines the columns and expressions that will appear in the result set, and can include the following data:

- **Predict** or **PredictOnly** columns from the mining model.
- Any column from the input data that is used to create the predictions.
- Functions that return a column of data.

The **FROM** <*model*> **PREDICTION JOIN** element defines the source data to be used to create the prediction. For a singleton query, this is a series of values that are assigned to columns. For an empty prediction join, this is left empty.

The **ON** element maps the columns that are defined in the mining model to columns in an external dataset. You do not have to include this element if you are creating an empty prediction join query or a natural prediction join.

You can use the **WHERE** clause to filter the results of a prediction query. You can use a **TOP** or **ORDER BY** clause to select most likely predictions. For more information about using these clauses, see Understanding the Select Statement (DMX).

For more information about the syntax of a prediction statement, see SELECT FROM PREDICTION JOIN (DMX) and SELECT FROM <model> (DMX).

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Understanding the Select Statement (DMX)

Understanding the Select Statement

The SELECT statement is the basis for most queries that you create with Data Mining Extensions (DMX) in Microsoft SQL Server Analysis Services. It can perform many different kinds of tasks, such as browsing and predicting against data mining models.

Following are the tasks that you can complete by using the **SELECT** statement:

- Browse a data mining model. The schema rowset defines the structure of a model.
- Discover the possible values of a mining model column.
- Browse the cases that are assigned to nodes in a mining model, or browse representations of those cases.
- Perform predictions against a variety of input sources.
- Copy mining models.

Each of these tasks uses a different data domain. You define the data domain in the **FROM** clause of the statement. For example, if you are browsing the data mining model object, your data domain is the columns that are defined by the schema rowset. Conversely, if you browse the cases of the model, your data domain is the actual column names in the model, such as Gender, Bike Buyer, and so on. In the first case, you are looking at the metadata that is stored in the schema rowset that defines the model; in the second case, you are actually looking at values, or representations of the values, that were used to train the mining model.

Anything that is included in the expression list or in the **WHERE** clause must come from the data domain that is defined by the **FROM** clause.

SELECT Types

You use the clauses in the **SELECT** statement to define the type of task that you want to perform. You can perform the following categories of tasks:

- Predicting
- Browsing
- Copying
- Drillthrough

Predicting

You can perform predictions based on a mining model by using the following query types.

Query Type	Traits
SELECT FROM [NATURAL] PREDICTION JOIN	Returns a prediction that is created by joining the columns in the mining model to the columns of an internal data source. The domain for this query type is the predictable columns from the model and the columns from the input data source.
SELECT FROM <*model*>	Returns the most likely state of the predictable column, based only on the mining model. This query type is a shortcut for creating a prediction with an empty prediction join. The domain for this query type is the predictable columns from the model.

You can include any one of the browsing or predicting **SELECT** statements within the **FROM** and **WHERE** clauses of a prediction join **SELECT** statement. For more information about imbedding a select statement, see **SELECT FROM PREDICTION JOIN (DMX)**. For more information about prediction query types and structure, see Prediction Queries (DMX).

Back to Select Types

Browsing

You can browse the contents of a mining model by using the following query types.

Query Type	**Traits**
SELECT DISTINCT FROM <*model*>	Returns all the state values from the mining model for the specified column. The domain for this query type is the data mining model.
SELECT FROM <*model*>**.CONTENT**	Returns content that describes the mining model. The domain for this query type is the content schema rowset.
SELECT FROM <*model*>**.DIMENSION_CONTENT**	Returns content that describes the mining model. The domain for this query type is the content schema rowset.
SELECT FROM <*model*>**.PMML**	Returns the Predictive Model Markup Language (PMML) representation of the mining model, for algorithms that support this functionality. The domain for this query type is the PMML schema rowset.

Back to Select Types

Copying

You can copy a mining model and its associated mining structure into a new model, which you name within the statement, by using the following query type.

Query Type	**Traits**
SELECT INTO <*new model*>	Creates a copy of the mining model. The domain for this query type is the data mining model.

Back to Select Types

Drillthrough

You can browse the cases, or a representation of the cases, that were used to train the model, by using the following query types.

Query Type	Traits
SELECT FROM <*model*>**.CASES** or **SELECT FROM** <*model*>**.SAMPLE_CASES**	Returns cases, or a representation of cases, that were used to train the mining model. The domain for this query type is the data mining model.

Back to Select Types

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

General Prediction Functions

You can use the **SELECT** statement in Data Mining Extensions (DMX) to create different types of queries. A query can be used to return information about the mining model itself, to make new predictions, or alter the model by training it with new data. Analysis Services provides a variety of specialized functions that control the type of information that is returned in a query. By adding these functions to a DMX query, you can retrieve additional statistics or columns of data. However, each query type and each model type supports certain functions only.

Common Functions

You can use functions to extend the results that a mining model returns. You can use the following functions for any **SELECT** statement that returns a table expression:

BottomCount	RangeMin
BottomPercent	TopCount
Predict	TopPercent
RangeMax	TopSum
RangeMid	

In addition, the following functions are supported for almost all model types:

- Exists (DMX)
- IsDescendent
- IsTestCase
- IsTrainingCase
- Predict
- RangeMax
- RangeMid
- RangeMin
- StructureColumn (DMX)

Individual algorithms may support additional functions. For a list of the functions that are supported by each model type, see Querying Data Mining Models (Analysis Services - Data Mining).

Functions Specific to SELECT Syntax

The following table lists the functions that you can use for each type of **SELECT** statement.

For general information about functions in DMX, see Understanding the Select Statement (DMX).

Query type	**Supported functions**	**Remarks**
SELECT DISTINCT FROM <model>	RangeMin (DMX) RangeMid (DMX) RangeMax (DMX)	These functions can be used to provide maximum values, minimum values, and

Query type	Supported functions	Remarks
		means for any column that contains numeric data type, regardless of whether the column is continuous or has been discretized.

SELECT FROM <model>.CONTENT
or
SELECT FROM <model>.DIMENSION_CONTENT
	IsDescendant (DMX)	This function retrieves child nodes for the specified node in the model, and can be used, for example, to iterate through the nodes in the mining model content. The arrangement of the nodes in the mining model content depends on the model type. For information about the structure for each mining model type, see Mining Model Content (Analysis Services - Data Mining).
		If you have saved the mining model content as a dimension, you can also use other Multidimensional Expressions (MDX) functions that are avaialble for querying an attribute hierarchy.

SELECT FROM <model>.CASES	IsInNode (DMX)	The **Lag** function is

Query type	Supported functions	Remarks
	ClientSettingsGeneralFlag Class IsTrainingCase (DMX) IsTestCase (DMX)	supported only for time series models. The **IsTestCase** function is supported in models that are based on a structure that was created using the holdout option, to create a testing data set. If the model is not based on a structure with holdout test set, all cases are considered training cases.
SELECT FROM <model>.SAMPLE_CASES	IsInNode (DMX)	In this context, the **IsInNode** function returns a case that belongs to a set of idealized sample cases.
SELECT FROM <model>.PMML	Not applicable. Use XML query functions instead.	PMML representations are supported only for the following model types: Microsoft Decision Trees Microsoft Clustering
SELECT FROM <model> PREDICTION JOIN	Prediction functions that are specific to the algorithm that you use to build the model.	For a list of prediction functions for each model type, see Querying Data Mining Models (Analysis Services - Data Mining).
SELECT FROM <model>	Prediction functions that are specific to the algorithm that you use to build the	For a list of prediction functions for each model type,

Query type	Supported functions	Remarks
	model.	see Querying Data Mining Models (Analysis Services - Data Mining).

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Data Mining Extensions (DMX) Syntax Elements

In Microsoft SQL Server Analysis Services, you can use various syntax elements to define Data Mining Extensions (DMX) statements that you can use to build, manage, and work with data mining models. The following sections describe these syntax elements.

In This Section

Understanding the Select Statement (DMX)

Name objects such as mining models, mining structures, and columns.

Data Types (DMX)

Define the type of data that a mining model column contains.

Expressions (DMX)

Units of syntax that Analysis Services can resolve to single, or scalar, values, objects, or table values.

Operators (DMX)

Used with one or more simple DMX expressions to make more complex DMX expressions.

Functions (DMX)

An expression that takes zero or one or more input values and returns a scalar value or a table.

Comments (DMX)

Text elements that you can insert into DMX statements or scripts to explain the purpose of a statement. Analysis Services does not run comments.

Reserved Keywords (DMX)

Words that are reserved for DMX use that should not be used to name objects in a database.

Content Types (DMX)

Define the content that a mining structure column contains.

Distributions (DMX)

Defines the distribution of data within a column.

Usage (DMX)

Define how a mining model uses the columns that it contains.

Modeling Flags (DMX)

Define additional hints that the algorithm can use to process a mining model.

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Identifiers

All objects in Microsoft SQL Server Analysis Services must have an identifier. An object's name is its identifier. Servers, databases, and database objects such as data sources, data source views, cubes, dimensions, mining models, and so on have identifiers.

There are two classes of identifiers in Data Mining Extensions (DMX):

- Regular identifiers
- Delimited identifiers

An object identifier is created when you define the object. You then use the identifier to reference the object. Identifiers must be 100 characters or less.

Regular Identifiers

Regular identifiers in DMX comply with the Analysis Services rules for the format of identifiers. Regular identifiers in DMX do not require delimiters. Following is an example of a DMX statement that uses a regular, non-delimited identifier:

```
SELECT * FROM Clustering.CONTENT
```

Rules for Regular Identifiers

Following are the rules for the format of regular identifiers:

1. The first character of a regular identifier must be one of the following:
   - A letter as defined by the Unicode Standard 2.0. This includes Latin characters from a through z and from A through Z, and letter characters from other languages.
   - An underscore (_).
2. Subsequent characters can be:
   - Letters as defined in the Unicode Standard 2.0.
   - Decimal numbers from either Basic Latin or other national scripts.
   - An underscore (_).
3. The identifier must not be a DMX reserved word. Reserved words are case-insensitive in DMX. For more information, see Understanding the Select Statement (DMX).
4. The identifier cannot contain embedded spaces or special characters.

You must delimit with brackets any identifiers that do not comply with these rules when you use them in DMX statements.

Delimited Identifiers

Delimited identifiers are enclosed in brackets ([ ]). Following is an example of a DMX statement with a delimited identifier that complies with those rules.

```
SELECT * FROM [Marketing_Clusters].CONTENT
```

An identifier that does not comply with the rules for the format of regular identifiers must always be delimited. Following is an example of DMX statement with a delimited identifier that contains a space:

```
SELECT * FROM [Targeted Mailing].CONTENT
```

Use delimited identifiers in the following situations:

- When you use reserved words for object names or parts of object names.

  We recommend that you do not use reserved keywords as object names. Databases that you upgrade from earlier versions of Analysis Services may contain identifiers that include words that were not reserved in the earlier version of Analysis Services but that are reserved words forSQL Server 2005 Analysis Services. You can use a delimited identifier to refer to such an object until you can change the object's name.

- When you use characters that are not listed as qualified identifiers.

  In Analysis Services you can use any character in the current code page in a delimited identifier; however, indiscriminate use of special characters in an object name may make DMX statements difficult to read and maintain.

Rules for Delimited Identifiers

Following are the rules for the format of delimited identifiers:

1. Delimited identifiers can contain the same number of characters as regular identifiers (from 1 through 100 characters, not including the delimiter characters).
2. The body of an identifier can contain any combination of characters that are used in the current code page, including the delimiting characters themselves. If the body of the identifier itself contains delimiting characters, special handling is required:
   - If the body of the identifier contains a left bracket ([), no additional handling is required.
   - If the body of the identifier contains a right bracket (]), you must specify two right brackets (]]) to represent it within the code page.

Delimiting Identifiers with Multiple Parts

When you use qualified object names, you may have to delimit more than one of the identifiers that make up the object name. You must delimit each identifier individually.

See Also

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Syntax Elements

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Data Types

When you use Data Mining Extensions (DMX) to define a new mining model in Microsoft SQL Server Analysis Services, you must provide a data type for each column in the model. The data type describes the data that the data mining algorithm will use when it builds the model.

Data types depend on the algorithm. Each algorithm determines which data types are supported and how they are used. Microsoft algorithms support the following data types:

- Text
- Long
- Boolean
- Double
- Date

For more information about the data types that Analysis Services supports, see Understanding the Select Statement (DMX).

See Also

Data Mining Algorithms

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Expressions

In Data Mining Extensions (DMX), an expression is a combination of identifiers, values, and operators that Microsoft SQL Server Analysis Services can evaluate to obtain a result.

A DMX expression can be simple or complex. A simple expression can be one of the following:

Constant

A constant is symbol that represents a single, specific value. A constant can be a string, or a numeric or date value. You must use single quotation marks (') to delimit string and date constants.

Scalar Function

A scalar function returns a single value.

Non-Scalar Function

A non-scalar function returns a table.

Object Identifier

Object identifiers are considered to be simple expressions in DMX.

To build complex expressions, you can use operators to combine these expressions. For more information about operators, see Understanding the Select Statement (DMX).

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Operators

You can use Data Mining Extensions (DMX) operators to perform arithmetic, comparison, concatenation, and logical operations in a query in Microsoft SQL Server Analysis Services.

Analysis Services uses operators to perform the following actions:

- Search for values or objects that meet a specific condition.
- Implement a decision between values or expressions.

DMX uses several categories of operators, described in the following sections. For additional information about individual operators, see Understanding the Select Statement (DMX).

Operator category	**Type of operation**
Arithmetic Operators (DMX)	Perform addition, subtraction, multiplication, or division.
Comparison Operators (DMX)	Compare one value against another value or an expression.
Logical Operators (DMX)	Test for the truth of a condition, such as AND, OR, or NOT.
Unary Operators (DMX)	Perform an operation on a single operand.

You can use operators to combine smaller expressions in DMX into more complex expressions. In complex expressions, the operators are evaluated in order based on the Analysis Services definition of operator precedence. Operators that have higher precedence are performed before operators that have lower precedence. For more information about expressions, see Expressions (DMX).

When you combine simple expressions to form a complex expression, the data type of the resulting expression is determined by combining the rules for the operators with the rules for data type precedence. If the result is a character or a Unicode value, Analysis Services determines the collation of the result by combining the rules for the operators with the rules for collation precedence. There are also rules that determine the precision, scale, and length of the result based on the precision, scale, and length of the simple expressions.

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Arithmetic Operators

You can use arithmetic operators in Data Mining Extensions (DMX) for arithmetic computations in Microsoft SQL Server Analysis Services, including addition, subtraction, multiplication, and division.

The following table identifies the arithmetic operators that DMX supports.

Operator	Description
Understanding the Select Statement (DMX)	Adds two numbers together.
- (Subtract) (DMX)	Subtracts one number from another number.
* (Multiply) (DMX)	Multiplies one number by another number.
/ (Divide) (DMX)	Divides one number by another number.

The following rules determine the order of precedence for arithmetic operators in a DMX expression:

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

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Expressions (DMX)

Mapping Functions to Query Types (DMX)

Operators (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Comparison Operators

You can use comparison operators with scalar data in any Data Mining Extensions (DMX) expression in Microsoft SQL Server Analysis Services. Comparison operators evaluate to a Boolean data type; they return TRUE or FALSE based on the outcome of the tested condition.

The following table identifies the comparison operators that DMX supports.

Operator	Description
Understanding the Select Statement (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the argument on the left is less than the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
> (Greater Than) (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the

Operator	Description
	argument on the left is greater than the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
= (Equal To) (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the argument on the left is equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
<> (Not Equal To) (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the argument on the left is not equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
<= (Less Than or Equal To) (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the argument on the left is less than or equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
>= (Greater Than or Equal To) (DMX)	For arguments that evaluate to a non-null value, returns TRUE if the value of the argument on the left is greater than or equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.

You can also use comparison operators in DMX statements and functions to look for a condition.

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference
DMX Operator Reference
DMX Statement Reference
DMX Syntax Conventions
DMX Syntax Elements
Expressions (DMX)
Mapping Functions to Query Types (DMX)
Operators (DMX)
Prediction Queries (DMX)
Understanding the Select Statement (DMX)

Logical Operators

You can use logical operators in Data Mining Extensions (DMX) expressions to evaluate values and to return a Boolean value in Microsoft SQL Server Analysis Services.

The following table identifies the logical operators that DMX supports.

Operator	Description
Understanding the Select Statement (DMX)	Performs a logical conjunction on two numeric expressions.
NOT (DMX)	Performs a logical negation on a numeric expression.
OR (DMX)	Performs a logical disjunction on two numeric expressions.

See Also

Data Mining Extensions (DMX) Reference
DMX Function Reference
DMX Operator Reference
DMX Statement Reference
DMX Syntax Conventions
DMX Syntax Elements
Expressions (DMX)
Mapping Functions to Query Types (DMX)
Operators (DMX)
Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Unary Operators

Unary operators perform an operation on a single operand, such as returning the negative or positive value of a numeric expression.

The following table identifies the unary operators that DMX supports.

Operator	Description
Understanding the Select Statement (DMX)	Returns the negative value of a numeric expression.
+ (Positive) (MDX)	Returns the positive value of a numeric expression.

See Also

Data Mining Extensions (DMX) Reference
DMX Function Reference
DMX Operator Reference
DMX Statement Reference
DMX Syntax Conventions
DMX Syntax Elements
Expressions (DMX)
Mapping Functions to Query Types (DMX)
Operators (DMX)
Prediction Queries (DMX)
Understanding the Select Statement (DMX)

Functions

When you use Data Mining Extensions (DMX) to query objects in Microsoft SQL Server Analysis Services, you can use functions to return more information than just the values in the columns in the data mining model or input dataset. For example, you can use DMX queries to return not only the prediction value of a column, but also the probability that the prediction is correct. You can use not only DMX functions, but also functions from Microsoft Visual Basic for Applications (VBA), Microsoft Excel, and stored procedures.

DMX Functions

You can use DMX functions to perform the following tasks:

- Return predictions.
- Return statistics about a prediction such as the probability and support.
- Filter your query results.
- Reorder a table expression.

Most DMX functions return a scalar value, such as the support for a prediction, but some return a tabular result. For example, the **PredictHistogram** function returns a table that contains the support and probability for each state of the specified predictable column. The results are displayed as a new tabular column.

For More Information: Understanding the Select Statement (DMX), DMX Function Reference

Visual Basic for Applications (VBA) and Excel Functions

In addition to DMX functions, you can also call a variety of VBA and Excel functions from DMX statements. For example, you can use the **lCase** function to modify how the Attribute_Name column in the TM_Decision_Tree model content is displayed. This is shown in the following code sample.

```
SELECT lCase([Attribute_Name])

FROM [TM_Decision_Tree].CONTENT
```

If the same function exists in both VBA and Excel, you must prefix the function name in your DMX statement with either **VBA** or **Excel**. For example, you would use `VBA!Log` or `Excel!Log`. If the VBA or Excel function that you want to use also exists in DMX or Multidimensional Expressions (MDX), or if the function contains a dollar sign character ($), you must use square brackets ([]) to escape the function. For example, the function call would be `[VBA!Format]`.

Stored Procedures

You can use common language runtime programming languages to create stored procedures that extend the functionality of DMX. For example, a regression tree mining model returns coefficients, such as , , and so on, that describe the regression equation, but the model does not return the equation itself, such as . However, you can write a stored procedure that uses the data mining model object to navigate the content schema, and to return the regression equation as an output. Therefore, a DMX statement can return a list of the regression equations as part of a query result.

For More Information: Assemblies (Analysis Services)

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Comments

Comments in Data Mining Extensions (DMX) are text strings in program code that Microsoft SQL Server Analysis Services does not execute. Comments are also known as remarks. You can use comments to document code or to temporarily disable parts of a DMX statement or script when you are diagnosing the code.

Using comments to document your program code makes it easier to maintain the code in the future. You can use comments to record details such as the name of the program, the name of the developer who wrote the code, and the dates of major code changes. You can also use comments to describe complex calculations or to a programming method.

Following are basic guidelines for writing comments:

- You can use any alphanumeric characters or symbols within a comment. Analysis Services ignores all characters that are within a comment.
- There is no maximum length for a comment within a statement or script. A comment can be made up of one or more lines.

Analysis Services supports the following types of comment characters:

- **// (double forward slashes).** Use these comment characters to write a comment on the same line as code that is to be executed, or to write a comment on a line by itself. Analysis Services evaluates everything from the double forward slashes to the end of the line as part of the comment. To create a multiple-line comment, use the double forward slashes at the start of each line of comment. For more information about this comment character, see Understanding the Select Statement (DMX).
- **-- (double hyphens).** Use these comment characters to write a comment on the same line as code that is to be executed, or to write a comment on a line by itself. Analysis Services evaluates everything from the double hyphens to the end of the line as part of the comment. To create a multiple-line comment, use the double hyphens at the start of each line of comment. For more information about this comment character, see -- (Comment).
- **/* ... */ (forward slash-asterisk character pairs).** Use these comment characters to write a comment on the same line as the code that is to be executed, to write a comment on a line by itself, or even to write comments within executable code. Analysis Services evaluates everything from the open comment pair (/*) to the close comment pair (*/) as part of the comment. To create a multiple-line comment, start

the comment with the open-comment character pair (/*), and end the comment with the close-comment character pair (*/). No other comment characters should be included on any lines of the comment. For more information about this comment character, see /*...*/ (Comment).

See Also

Data Mining Extensions (DMX) Reference

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Reserved Keywords

Microsoft SQL Server 2005 Analysis Services (SSAS) reserves certain keywords for its exclusive use. These keywords cannot be used anywhere in Data Mining Extensions (DMX) statements except in the positions that Analysis Services defines in the DMX language reference. These restricted DMX keywords include the following members:

- All data definition statements listed in the topic, DMX Data Definition Statements.
- All data mining query functions listed in the topic, DMX Function Reference.
- All operators listed in the topic, DMX Operator Reference.
- Keywords that are defined in the Multidimensional Expressions (MDX) query language and are included as part of a DMX statement.
- Keywords that are defined in the OLE DB for Data Mining specification and are included as part of a DMX statement.

When you name objects in a database, we recommend that you use a naming convention that avoids using reserved keywords.

If your database does contain names that match reserved keywords, you must use delimited identifiers when you refer to those objects. For more information, see Identifiers.

See Also

Data Mining Extensions (DMX) Reference

DMX Statement Reference

DMX Syntax Conventions

DMX Syntax Elements

Understanding the Select Statement (DMX)

Content Types

Data mining algorithms require additional information beyond the data type to function correctly, such as the content type. The content type helps the algorithm determine how to work with the data in the column.

Each algorithm supports specific content types. For example, the Microsoft Naive Bayes algorithm cannot use continuous columns. To use a continuous column in a Microsoft Naive Bayes model, you must discretize the data in the column. Some algorithms require certain content types in order to function correctly. For example, the Microsoft Time Series algorithm requires a key time column to identify the time over which the data was collected.

For a complete description of the content types that Analysis Services supports, see Understanding the Select Statement (DMX).

See Also

Data Mining Algorithms

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Syntax Elements

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Distributions

In Microsoft SQL Server Analysis Services, you can define the content of columns in a mining structure, to affect how algorithms process the data in those columns when you create mining models. For some algorithms, it is useful to define the distribution of any continuous columns before you process the model, if the columns are known to contain common distributions of values. If you do not define the distributions, the resulting mining models may produce less accurate predictions than if the distributions were defined, because the algorithms will have less information from which to interpret the data.

Microsoft data mining algorithms support the following distribution types:

NORMAL

The values for the continuous column form a histogram with a normal Gaussian distribution.

Log Normal

The values for the continuous column form a histogram, where the logarithm of the values is normally distributed.

UNIFORM

The values for the continuous column form a flat curve, in which all values are equally likely.

For more information about Microsoft data mining algorithms, see Understanding the Select Statement (DMX). Third-party algorithm providers may support additional distribution types. To determine which distribution types an algorithm supports, use the **SUPPORTED_DISTRIBUTION_FLAGS** schema rowset.

For more information about distribution types, see Column Distributions.

See Also

Content Types

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Syntax Elements

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Usage

When you use Data Mining Extensions (DMX) to define a new data mining model in Microsoft SQL Server Analysis Services, you must specify how the data mining algorithm that builds the model will use each column. You can specify a column as one of the following types:

- **Key**
- **Key Sequence**
- **Key Time**
- **Predict**
- **PredictOnly**

Columns that are left unspecified in DMX are treated as input columns.

To process a model correctly, the algorithm must know which column is the key column that uniquely identifies each row, which column is the target column for creating predictions if you are creating a predictable model, and which columns to use as input columns to create the relationships that predict the target column.

Columns that are specified as the **Predict** type are used as both input and output columns. Columns that are specified as **PredictOnly** are only used as output columns. Specific algorithms may treat Predict columns differently.

For more information about the column usage types that Analysis Services supports, see Understanding the Select Statement (DMX).

See Also

Data Mining Algorithms

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Syntax Elements

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Modeling Flags

You can use modeling flags in Analysis Services to provide additional information to a data mining algorithm about the data that is defined in a case table. The algorithm can use this information to build a more accurate data mining model. You can define modeling flags on both mining structure columns and mining model columns.

Analysis Services supports the following modeling flags:

NOT NULL

The values for the attribute column should never contain a null value. An error will result if Analysis Services encounters a null value for this attribute column during the model training process. This flag is defined on a mining structure column.

REGRESSOR

Indicates that the algorithm can use the specified column in the regression formula of regression algorithms. This flag is supported by the Microsoft Linear Regression and Microsoft Decision Trees algorithms, and is defined on a mining model column.

MODEL_EXISTENCE_ONLY

The values for the attribute column are less important than the presence of the attribute. This flag is defined on a mining model column.

Third-party algorithms may support additional modeling flags. To determine which modeling flags an algorithm supports, use the **SUPPORTED_MODELING_FLAGS** schema rowset. You can also query the mining services on the server to determine which modeling flags are supported for a particular algorithm. For example, the following query returns the modeling flags are supported for the Microsoft Linear Regression algorithm on the current server:

```
SELECT SUPPORTED_MODELING_FLAGS

FROM $SYSTEM.DMSCHEMA_MINING_SERVICES

WHERE SERVICE_NAME = 'Microsoft_Linear_Regression'
```

Expected results:

NOT NULL,REGRESSOR

Specifying Modeling Flags on a Mining Model

For examples of the syntax that Analysis Services supports for specifying a flag on a mining structure column, see CREATE MINING STRUCTURE (DMX).

For an example of the syntax for specifying a modeling flga on a mining model column, see ALTER MINING STRUCTURE (DMX).

For more information about working with mining model columns, see Mining Model Columns.

See Also

Data Mining Algorithms

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Syntax Elements

DMX Function Reference

DMX Operator Reference

DMX Statement Reference

DMX Syntax Conventions

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

Data Mining Extensions (DMX) Statement Reference

Working with data mining models in Microsoft SQL Server Analysis Services involves the following primary tasks:

- Creating mining structures and mining models
- Processing mining structures and mining models
- Deleting or dropping mining structures or mining models
- Copying mining models
- Browsing mining models
- Predicting against mining models

You can use Data Mining Extensions (DMX) statements to perform each of these tasks programmatically.

Creating mining structures and mining models

Use the Data Mining Extensions (DMX) Syntax Elements statement to add a new mining structure to a database. You can then use the ALTER MINING STRUCTURE statement to add mining models to the mining structure.

Use the CREATE MINING MODEL statement to build a new mining model and associated mining structure.

Processing mining structures and mining models

Use the INSERT INTO statement to process a mining structure and mining model.

Deleting or dropping mining structures or mining models

Use the DELETE statement to remove all the trained data from a mining model or mining structure. Use the DROP MINING STRUCTURE or DROP MINING MODEL statements to completely remove a mining structure or mining model from a database.

Copying mining models

Use the SELECT INTO statement to copy the structure of an existing mining model into a new mining model and to train the new model with the same data.

Browsing mining models

Use the SELECT statement to browse the information that the data mining algorithm calculates and stores in the data mining model during model training. Much like with Transact-SQL, you can use several clauses with the **SELECT** statement, to extend its power. These clauses include DISTINCT FROM <model>, FROM <model>.CASES, FROM <model>.SAMPLE_CASES, FROM <model>.CONTENT and FROM <model>.DIMENSION_CONTENT.

Predicting against mining models

Use the PREDICTION JOIN clause of the **SELECT** statement to create predictions that are based on an existing mining model.

You can also import and export models by using the IMPORT and EXPORT statements.

These tasks fall into two categories, data definition statements and data manipulation statements, which are described in the following table.

Topic	Description
DMX Data Definition Statements	Part of the data definition language (DDL). Used to define a new mining model (including training) or to drop an existing mining model from a database.
DMX Data Manipulation Statements	Part of the data manipulation language (DML). Used to work with existing mining models, including browsing a model or creating predictions.

See Also

Data Mining Extensions (DMX) Function Reference

Data Mining Extensions (DMX) Operator Reference

Data Mining Extensions (DMX) Syntax Conventions

Data Mining Extensions (DMX) Syntax Elements

Data Mining Extensions (DMX) Data Definition Statements

The following table lists the statements that are part of the data mining data definition language (DDL) in Data Mining Extensions (DMX).

Statement	Description
Data Mining Extensions (DMX) Statement Reference	Creates a new mining structure in the database.
ALTER MINING STRUCTURE (DMX)	Adds a mining model to an existing mining structure.
CREATE MINING MODEL (DMX)	Creates a new mining structure and mining

Statement	Description
	model in the database.
DROP MINING MODEL (DMX)	Deletes a mining model from the database.
DROP MINING STRUCTURE (DMX)	Deletes a mining structure from the database.
EXPORT (DMX)	Exports a mining model or mining structure and associated objects into a .abf file.
IMPORT (DMX)	Imports a mining model or mining structure and associated objects from a .abf file.
SELECT INTO (DMX)	Creates a copy of an existing mining model.

See Also

DMX Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

CREATE MINING STRUCTURE

Creates a new mining structure in a database and optionally defines training and testing partitions. After you have created the mining structure, you can use the ALTER MINING STRUCTURE (DMX) statement to add models to the mining structure.

Syntax

```
CREATE [SESSION] MINING STRUCTURE <structure>
(
   [(<column definition list>)]
)
[WITH HOLDOUT (<holdout-specifier> [OR <holdout-specifier>])]
[REPEATABLE(<holdout seed>)]
<holdout-specifier>::=  <holdout-maxpercent> PERCENT | <holdout-maxcases> CASES
```

Arguments

structure

A unique name for the structure.

column definition list

A comma-separated list of column definitions.

holdout-maxpercent

An integer between 1 and 100 that indicates the percentage of data to set aside for testing.

holdout-maxcases

An integer that indicates the maximum number of cases to use for testing.

If the value specified for max cases is larger than the number of input cases, all input cases are used for testing and a warning will be raised.

Note

If both percentage and maximum number of cases is specified, the smaller of the two limits is used.

holdout seed

An integer used as the seed to start partitioning data.

If set to 0, the hash of the mining structure ID is used as the seed.

Note

You should specify a seed if you need to ensure that a partition can be reproduced.

Default: REPEATABLE(0)

Remarks

You define a mining structure by specifying a list of columns, optionally specifying hierarchical relationships between the columns, and then optionally partitioning the mining structure into training and testing data sets.

The optional SESSION keyword indicates that the structure is a temporary structure that you can use only for the duration of the current session. When the session is terminated, the structure, and any models based on the structure, will be deleted. To create temporary mining structures and models, you must first set the database property, **AllowSessionMiningModels**. For more information, see Data Mining Properties.

Column Definition List

You define a mining structure by including the following information for each column in the column definition list:

- Name (mandatory)
- Data type (mandatory)
- Distribution
- List of modeling flags
- Content type (mandatory)

- Relationship to an attribute column (mandatory only if it applies), indicated by the RELATED TO clause

Use the following syntax for the column definition list to define a single column:

```
<column name>    <data type>    [<Distribution>]    [<Modeling Flags>]
<Content Type>    [<column relationship>]
```

Use the following syntax for the column definition list to define a nested table column:

```
<column name>    TABLE    ( <column definition list> )
```

For a list of the data types, content types, column distributions, and modeling flags that you can use to define a structure column, see the following topics:

- Data Types (Data Mining)
- Content Types (Data Mining)
- Column Distributions
- Modeling Flags (Data Mining)

You can define multiple modeling flags values for a column. However, you can have only one content type and one data type for a column.

Column Relationships

You can add a clause to any column definition statement to describe the relationship between two columns. Analysis Services supports the use of the following <column relationship> clause.

RELATED TO

Indicates a value hierarchy. The target of a RELATED TO column can be a key column in a nested table, a discretely-valued column in the case row, or another column with a RELATED TO clause, which indicates a deeper hierarchy.

Holdout Parameters

When you specify holdout parameters, you create a partition of the structure data. The amount that you specify for holdout is reserved for testing, and the remaining data is used for training. By default, if you create a mining structure by using SQL Server Data Tools (SSDT), a holdout partition is created for you that contains 30 percent testing data and 70 percent training data. For more information, see Partitioning Data into Training and Testing Sets (Analysis Services - Data Mining).

If you create a mining structure by using Data Mining Extensions (DMX), you must manually specify that a holdout partition be created.

Note
The **ALTER MINING STRUCTURE** statement does not support holdout.

You can specify up to three holdout parameters. If you specify both a maximum number of holdout cases and a holdout percentage, a percentage of cases are reserved until the maximum cases limit is reached. You specify the percentage of holdout as an integer followed by the **PERCENT** keyword, and specify the maximum number of cases as an integer followed by the **CASES** keyword. You can combine the conditions in any order, as shown in the following examples:

```
WITH HOLDOUT (20 PERCENT)

WITH HOLDOUT (2000 CASES)

WITH HOLDOUT (20 PERCENT OR 2000 CASES)

WITH HOLDOUT (2000 CASES OR 20 PERCENT)
```

The holdout seed controls the starting point of the process that randomly assigns cases to either the training or testing data sets. By setting a holdout seed, you can ensure that the partition can be repeated. If you do not specify a holdout seed, Analysis Services uses the name of the mining structure to create a seed. If you rename the structure, the seed value will change. The holdout seed parameter can be used with either or both of the other holdout parameters.

Note

Because the partition information is cached with the training data, to use holdout, you must ensure that the **CacheMode** property of the mining structure is set to **KeepTrainingData**. This is the default setting in Analysis Services for new mining structures. Changing the **CacheMode** property to **ClearTrainingCases** on an existing mining structure that contains a holdout partition will not affect any mining models that have been processed. However, if **T:Microsoft.AnalysisServices.MiningStructureCacheMode** is not set to **KeepTrainingData**, holdout parameters will have no effect. This means that all the source data will be used for training and no test set will be available. The definition of the partition is cached with the structure; if you clear the cache of training cases, you also clear the cache of test data, and the definition of the holdout set.

Examples

The following examples demonstrate how to create a mining structure with holdout by using DMX.

Example 1: Adding a Structure with No Training Set

The following example creates a new mining structure called `New Mailing` without creating any associated mining models, and without using holdout. To learn how to add a mining model to the structure, see ALTER MINING STRUCTURE (DMX).

```
CREATE MINING STRUCTURE [New Mailing]

(

    CustomerKey LONG KEY,

    Gender TEXT DISCRETE,

    [Number Cars Owned] LONG DISCRETE,

    [Bike Buyer] LONG DISCRETE
)
```

Example 2: Specifying Holdout Percentage and Seed

The following clause can be added after the column definition list to define a data set that can be used for testing all mining models associated with the mining structure. The statement will create a test set that is 25 percent of the total input cases, without a limit on the maximum number of cases. 5000 is used as the seed for creating the partition. When you specify a seed, the same cases will be chosen for the test set each time you process the mining structure, so long as the underlying data does not change.

```
CREATE MINING STRUCTURE [New Mailing]
(
    CustomerKey LONG KEY,
    Gender TEXT DISCRETE,
    [Number Cars Owned] LONG DISCRETE,
    [Bike Buyer] LONG DISCRETE
)
WITH HOLDOUT(25 PERCENT) REPEATABLE(5000)
```

Example 3: Specifying Holdout Percentage and Max Cases

The following clause will create a test set that contains either 25 percent of the total input cases, or 2000 cases, whichever is less. Because 0 is specified as the seed, the name of the mining structure is used to create the seed that is used to begin sampling of the input cases.

```
CREATE MINING STRUCTURE [New Mailing]
(
    CustomerKey LONG KEY,
    Gender TEXT DISCRETE,
    [Number Cars Owned] LONG DISCRETE,
    [Bike Buyer] LONG DISCRETE
)
WITH HOLDOUT(25 PERCENT OR 2000 CASES) REPEATABLE(0)
```

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

ALTER MINING STRUCTURE

Creates a new mining model that is based on an existing mining structure. When you use the **ALTER MINING STRUCTURE** statement to create a new mining model, the structure must already exist. In contrast, when you use the statement, CREATE MINING MODEL (DMX), you create a model and automatically generate its underlying mining structure at the same time.

Syntax

```
ALTER MINING STRUCTURE <structure>
ADD MINING MODEL <model>
(
   <column definition list>
  [(<nested column definition list>) [WITH FILTER (<nested filter criteria>)]]
)
USING <algorithm> [(<parameter list>)]
[WITH DRILLTHROUGH]
[,FILTER(<filter criteria>)]
```

Arguments

structure

The name of the mining structure to which the mining model will be added.

model

A unique name for the mining model.

column definition list

A comma-separated list of column definitions.

nested column definition list

A comma-separated list of columns from a nested table, if applicable.

nested filter criteria

A filter expression that is applied to the columns in a nested table.

algorithm

The name of a data mining algorithm, as defined by the provider.

Note

A list of the algorithms supported by the current provider can be retrieved by using DMSCHEMA_MINING_SERVICES Rowset. To view the algorithms supported in the current instance of Analysis Services, see Data Mining Properties.

parameter list

Optional. A comma-separated list of provider-defined parameters for the algorithm.

filter criteria

A filter expression that is applied to the columns in the case table.

Remarks

If the mining structure contains composite keys, the mining model must include all the key columns that are defined in the structure.

If the model does not require a predictable column, for example, models that are built by using the Microsoft Clustering and Microsoft Sequence Clustering algorithms, you do not have to include a column definition in the statement. All the attributes in the resulting model will be treated as inputs.

In the **WITH** clause that applies to the case table, you can specify options for both filtering and drillthrough:

- Add the **FILTER** keyword and a filter condition. The filter applies to the cases in the mining model.
- Add the **DRILLTHROUGH** keyword to enable users of the mining model to drill down from model results to the case data. In Data Mining Extensions (DMX), drillthrough can be enabled only when you create the model.

To use both case filtering and drillthrough, you combine the keywords in a single **WITH** clause by using the syntax shown in the following example:

```
WITH DRILLTHROUGH, FILTER(Gender = 'Male')
```

Column Definition List

You define the structure of a model by specifying a column definition list that includes the following information for each column:

- Name (mandatory)
- Alias (optional)
- Modeling flags
- Prediction request, which indicates to the algorithm whether the column contains a predictable value, indicated by the **PREDICT** or **PREDICT_ONLY** clause

Use the following syntax for the column definition list to define a single column:

```
<structure column name>  [AS <model column name>]  [<modeling flags>]
[<prediction>]
```

Column Name and Alias

The column name that you use in the column definition list must be the name of the column as it is used in the mining structure. However, you can optionally define an alias to represent the structure column in the mining model. You can also create multiple column definitions for the same structure column, and assign a different alias and prediction usage to each copy of the column. By default, the structure column name is used if you do not define an alias. For more information, see How to: Create an Alias for a Model Column.

For nested table columns, you specify the name of the nested table, specify the data type as **TABLE**, and then provide the list of nested columns to include in the model, enclosed in parentheses.

You can define a filter expression that is applied to the nested table by affixing a filter criteria expression after the nested table column definition.

Modeling Flags

Analysis Services supports the following modeling flags for use in mining model columns:

Note
The NOT_NULL modeling flag applies to the mining structure column. For more information, see CREATE MINING STRUCTURE (DMX).

Term	Definition
REGRESSOR	Indicates that the algorithm can use the specified column in the regression formula of regression algorithms.
MODEL_EXISTENCE_ONLY	Indicates that the values for the attribute column are less important than the presence of the attribute.

You can define multiple modeling flags for a column. For more information about how to use modeling flags, see Modeling Flags (DMX).

Prediction Clause

The prediction clause describes how the prediction column is used. The following table lists the possible clauses.

PREDICT	This column can be predicted by the model, and its values can be used as input to predict the value of other predictable columns.
PREDICT_ONLY	This column can be predicted by the model, but its values cannot be used in input cases to predict the value of other predictable columns.

Filter Criteria Expressions

You can define a filter that restricts the cases that are used in the mining model. The filter can be applied to either the columns in the case table or the rows in the nested table, or to both.

Filter criteria expressions are simplified DMX predicates, similar to a WHERE clause. Filter expressions are restricted to formulas that use basic mathematical operators, scalars, and column names. The exception is the EXISTS operator; it evaluates to true if at least one row is returned for the subquery. Predicates can be combined by using the common logical operators: AND, OR, and NOT.

For more information about filters used with mining models, see Creating Filters for Mining Models (Analysis Services - Data Mining).

Note

Columns in a filter must be mining structure columns. You cannot create a filter on a model column or an aliased column.

For more information about DMX operators and syntax, see Data Mining Extensions (DMX) Statement Reference.

Parameter Definition List

You can adjust the performance and functionality of a model by adding algorithm parameters to the parameter list. The parameters that you can use depend on the algorithm that you specify in the USING clause. For a list of parameters that are associated with each algorithm, see Data Mining Algorithms.

The syntax of the parameter list is as follows:

```
[<parameter> = <value>, <parameter> = <value>,…]
```

Example 1: Add a Model to a Structure

The following example adds a Naive Bayes mining model to the **New Mailing** mining structure and limits the maximum number of attribute states to 50.

```
ALTER MINING STRUCTURE [New Mailing]

ADD MINING MODEL [Naive Bayes]

(

    CustomerKey,

    Gender,

    [Number Cars Owned],

    [Bike Buyer] PREDICT

)

USING Microsoft_Naive_Bayes (MAXIMUM_STATES = 50)
```

Example 2: Add a Filtered Model to a Structure

The following example adds a mining model, `Naive Bayes Women`, to the **New Mailing** mining structure. The new model has the same basic structure as the mining model that was added in example 1; however, this model restricts the cases from the mining structure to female customers over the age of 50.

```
ALTER MINING STRUCTURE [New Mailing]
ADD MINING MODEL [Naive Bayes Women]
(
    CustomerKey,
    Gender,
    [Number Cars Owned],
    [Bike Buyer] PREDICT
)
USING Microsoft_Naive_Bayes
WITH FILTER([Gender] = 'F' AND [Age] >50)
```

Example 3: Add a Filtered Model to a Structure with a Nested Table

The following example adds a mining model to a modified version of the market basket mining structure. The mining structure used in the example has been modified to add a **Region** column, which contains attributes for customer region, and an **Income Group** column, which categorizes customer income by using the values **High**, **Moderate**, or **Low**.

The mining structure also includes a nested table that lists the items that the customer has purchased.

Because the mining structure contains a nested table, you can define a filter on the case table, the nested table, or both. This example combines a case filter and nested row filter to restrict the cases to wealthy European customers who purchased one of the road tire models.

```
ALTER MINING STRUCTURE [Market Basket with Region and Income]
ADD MINING MODEL [Decision Trees]
(
    CustomerKey,
    Region,
    [Income Group],
    [Product] PREDICT (Model)
WITH FILTER (EXISTS (SELECT * FROM [v Assoc Seq Line Items] WHERE
 [Model] = 'HL Road Tire' OR
 [Model] = 'LL Road Tire' OR
```

```
  [Model] = 'ML Road Tire' )
)
) WITH FILTER ([Income Group] = 'High' AND [Region] = 'Europe')
USING Microsoft_Decision Trees
```

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

CREATE MINING MODEL

Creates both a new mining model and a mining structure in the database. You can create a model either by defining the new model in the statement, or by using the Predictive Model Markup Language (PMML). This second option is for advanced users only.

The mining structure is named by appending "_structure" to the model name, which ensures that the structure name is unique from the model name.

To create a mining model for an existing mining structure, use the ALTER Mining STRUCTURE statement.

Syntax

```
CREATE [SESSION] MINING MODEL <model>
(
   [(<column definition list>)]
)
USING <algorithm> [(<parameter list>)] [WITH DRILLTHROUGH]
CREATE MINING MODEL <model> FROM PMML <xml string>
```

Arguments

model

A unique name for the model.

column definition list

A comma-separated list of column definitions.

algorithm

The name of a data mining algorithm, as defined by the current provider.

Note

A list of the algorithms supported by the current provider can be retrieved by using DMSCHEMA_MINING_SERVICES Rowset. To view the algorithms supported in the current instance of Analysis Services, see Data Mining Properties.

parameter list

Optional. A comma-separated list of provider-defined parameters for the algorithm.

XML string

(For advanced use only.) An XML-encoded model (PMML). The string must be enclosed in single quotation marks (').

The **SESSION** clause lets you create a mining model that is automatically removed from the server when the connection closes or the session times out. **SESSION** mining models are useful because they do not require the user to be a database administrator, and they only use disk space for as long as the connection is open.

The **WITH DRILLTHROUGH** clause enables drill through on the new mining model. Drillthrough can only be enabled when you create the model. For some model types, drillthrough is required in order to browse the model in the custom viewer. Drillthrough is not required for prediction or for browsing the model by using the Microsoft Generic Content Tree Viewer.

The **CREATE MINING MODEL** statement creates a new mining model that is based on the column definition list, the algorithm, and the algorithm parameter list.

Column Definition List

You define the structure of a model that uses the column definition list by including the following information for each column:

- Name (mandatory)
- Data type (mandatory)
- Distribution
- List of modeling flags
- Content type (mandatory)
- Prediction request, which indicates to the algorithm to predict this column, indicated by the **PREDICT** or **PREDICT_ONLY** clause
- Relationship to an attribute column (mandatory only if it applies), indicated by the **RELATED TO** clause

Use the following syntax for the column definition list, to define a single column:

```
<column name>    <data type>    [<Distribution>]    [<Modeling Flags>]
<Content Type>    [<prediction>]    [<column relationship>]
```

Use the following syntax for the column definition list, to define a nested table column:

```
<column name>    TABLE    [<prediction>] ( <non-table column definition
list> )
```

Except for modeling flags, you can use no more than one clause from a particular group to define a column. You can define multiple modeling flags for a column.

For a list of the data types, content types, column distributions, and modeling flags that you can use to define a column, see the following topics:

- Data Types (Data Mining)
- Content Types (Data Mining)
- Column Distributions
- Modeling Flags (Data Mining)

You can add a clause to the statement to describe the relationship between two columns. Analysis Services supports the use of the following <Column relationship> clause.

RELATED TO

This form indicates a value hierarchy. The target of a RELATED TO column can be a key column in a nested table, a discretely-valued column in the case row, or another column with a RELATED TO clause, which indicates a deeper hierarchy.

Use a prediction clause to describe how the prediction column is used. The following table describes the two possible clauses.

<prediction> clause	Description
PREDICT	This column can be predicted by the model, and it can be supplied in input cases to predict the value of other predictable columns.
PREDICT_ONLY	This column can be predicted by the model, but its values cannot be used in input cases to predict the value of other predictable columns.

Parameter Definition List

You can use the parameter list to adjust the performance and functionality of a mining model. The syntax of the parameter list is as follows:

```
[<parameter> = <value>, <parameter> = <value>,…]
```

For a list of the parameters that are associated with each algorithm, see Data Mining Algorithms.

Remarks

If you want to create a model that has a built-in testing data set, you should use the statement CREATE MINING STRUCTURE followed by ALTER MINING STRUCTURE. However, not all model types support a holdout data set. For more information, see CREATE MINING STRUCTURE (DMX).

For a walkthrough of how to create a mining model by using the CREATEMODEL statement, see Time Series Prediction DMX Tutorial.

Naive Bayes Example

The following example uses the Microsoft Naive Bayes algorithm to create a new mining model. The Bike Buyer column is defined as the predictable attribute.

```
CREATE MINING MODEL [NBSample]
(
    CustomerKey LONG KEY,
    Gender TEXT DISCRETE,
    [Number Cars Owned] LONG DISCRETE,
    [Bike Buyer] LONG DISCRETE PREDICT
)
USING Microsoft_Naive_Bayes
```

Association Model Example

The following example uses the Microsoft Association algorithm to create a new mining model. The statement takes advantage of the ability to nest a table inside the model definition by using a table column. The model is modified by using the MINIMUM_PROBABILITY and MINIMUM_SUPPORT parameters.

```
CREATE MINING MODEL MyAssociationModel (
    OrderNumber TEXT KEY,
    [Products] TABLE PREDICT (
        [Model] TEXT KEY
    )
)
USING Microsoft_Association_Rules (Minimum_Probability = 0.1,
MINIMUM_SUPPORT = 0.01)
```

Sequence Clustering Example

The following example uses the Microsoft Sequence Clustering algorithm to create a new mining model. Two keys are used to define the model. The OrderNumber column is used as the case key, and specifies individual orders. The LineNumber column is used as the nested table key, and specifies the sequence in which items were added to an order.

```
CREATE MINING MODEL BuyingSequence (
    [Order Number] TEXT KEY,
    [Products] TABLE
     (
        [Line Number] LONG KEY SEQUENCE,
        [Model] TEXT DISCRETE PREDICT
    )
)
USING Microsoft_Sequence_Clustering
```

Time Series Example

The following example uses the Microsoft Times Series algorithm to create a new mining model by using the ARTxp algorithm. ReportingDate is the key column for the time series and ModelRegion is the key column for the data series. In this example, it is assumed that the periodicity of the data is every 12 months. Therefore, the PERIODICITY_HINT parameter is set to 12.

Note
You must specify the PERIODICITY_HINT parameter by using brace characters. Moreover, because the value is a string, it must be enclosed in single quotation marks: "{<numeric value>}".

```
CREATE MINING MODEL SalesForecast (
        ReportingDate DATE KEY TIME,
        ModelRegion TEXT KEY,
        Amount LONG CONTINUOUS PREDICT,
        Quantity LONG CONTINUOUS PREDICT
)
USING Microsoft_Time_Series (PERIODICITY_HINT = '{12}', FORECAST_METHOD
= 'ARTXP')
```

See Also

Data Mining Extensions (DMX) Data Definition Statements
Data Mining Extensions (DMX) Data Manipulation Statements
Data Mining Extensions (DMX) Statement Reference

DROP MINING STRUCTURE

Drops the specified mining structure from the database. All the mining models that are associated with the structure are also dropped from the database.

Syntax

DROP MINING STRUCTURE <structure>

Arguments

structure

A structure identifier.

Examples

The following example drops the New Mailing mining structure from the database.

```
DROP MINING STRUCTURE [New Mailing]
```

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

DROP MINING MODEL

Deletes a mining model from the database.

Syntax

DROP MINING MODEL <model >

Arguments

model

A model identifier.

Examples

The following sample code drops the mining model NBSample.

```
DROP MINING MODEL [NBSample]
```

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

EXPORT

Extracts a mining model or mining structure object from the server to an Analysis Services Backup File (.abf).

Syntax

EXPORT <**object type**> <**object name**>[, <**object name**>] [<**object type**> <**object name**>[, <**object name**]] TO <**filename**> [WITH DEPENDENCIES]

Arguments

object type

Optional. The type of the object to export (either mining model or mining structure).

object name

Optional. The name of the object to export.

filename

The name and location of the file to export as a string.

Remarks

If the statement specifies a mining model, the resultant file will also contain an associated mining structure. If the statement specifies **WITH DEPENDENCIES**, all objects required to process the object (for example, the data source and data source view) are included in the .abf file.

You must be a database or server administrator to export or import objects from a Microsoft SQL Server Analysis Services database.

Export Mining Structure Example

The following example exports the Targeted Mailing and Forecasting mining structures, and the Association mining model to a specific file location. Because the Association model is part of the Market Basket mining structure, the example also exports the Market Basket structure. Any other mining models that may exist as part of the Market Basket mining structure will not be exported because the Association model was exported using **MINING MODEL**, not **MINING STRUCTURE**.

```
EXPORT MINING STRUCTURE [Targeted Mailing], [Forecasting] MINING MODEL
Association TO 'C:\TM_NEW.abf'
```

Export Mining Model Example

The following example exports the Association mining model to a specified file location. Because the statement specifies **WITH DEPENDENCIES**, the data source and data source view objects are also included in the .abf file.

```
EXPORT MINING MODEL [Association] TO 'C:\Association_NEW.abf' WITH
DEPENDENCIES
```

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Manipulation Statements

DMX Statement Reference

IMPORT (DMX)

Exporting and Importing Data Mining Projects (Analysis Services - Data Mining)

IMPORT

Loads a mining model or mining structure from an Analysis Services Backup File (.abf) file onto the server.

Syntax

IMPORT FROM <**filename**>

Arguments

filename

A string containing the name and location of the file to import.

Remarks

If no objects are specified, the entire contents of the .dmb file will be loaded. If the .dmb file includes a database that does not exist on the server, the database will be created.

You must be a database or server administrator to export or import objects.

Import from File Example

The following example imports the entire contents of the file containing the association model and structure onto the current server.

```
IMPORT FROM 'C:\TEMP\Association_NEW.dmb'
```

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Manipulation Statements

DMX Statement Reference

EXPORT (DMX)

Exporting and Importing Data Mining Projects (Analysis Services - Data Mining)

SELECT INTO

Creates a new mining model that is built on the mining structure of an existing mining model. The **SELECT INTO** statement creates the new mining model by copying schema and other information that is not specific to the actual algorithm.

Syntax

SELECT INTO <**new model**>

USING <**algorithm**> [(<**parameter list**>)] [WITH DRILLTHROUGH[,] [FILTER(<expression>)]]

FROM <**existing model**>

Arguments

new model

A unique name for the new model that is being created.

algorithm

The provider-defined name of a data mining algorithm.

parameter list

Optional. A comma-separated list of provider-defined parameters for the algorithm.

expression

An expression that evaluates to a valid filter condition on the training data. For more information about expressions that can be used as filters, see Creating Filters for Mining Models (Analysis Services - Data Mining).

existing model

The name of the existing model to be copied.

Remarks

If the existing model is trained, the new model is automatically processed when this statement executes. Otherwise, the new model remains unprocessed.

The **SELECT INTO** statement works only if the structure of the existing model is compatible with the algorithm of the new model. Therefore, this statement is most useful for rapidly creating and testing models that are based on the same algorithm. If you change the algorithm type, the new algorithm must support the data type of each column that is in the existing model, or an error might occur when the model is processed,

The **WITH DRILLTHROUGH** clause enables drillthrough on the new mining model. Drillthrough can only be enabled when you create the model.

Example 1: Altering the Parameters of the Model

The following example creates a new mining model based on an existing mining model, `TM_Clustering`, which you create in the Basic Data Mining Tutorial. In the new model, the CLUSTER_COUNT parameter is modified so that a maximum of five clusters will exist in the new model. In contrast, the existing model uses the default value, which is 10.

```
SELECT * INTO [New_Clustering]

USING [Microsoft_Clustering] (CLUSTER_COUNT = 5)

FROM [TM Clustering]
```

Example 2: Adding a Filter to the Model

The following example creates a new mining model based on an existing mining model, and adds a filter on the model. The filter restricts the training data to only those customers who live in a particular region.

```
SELECT * INTO [Clustering Europe Region]

USING [Microsoft_Clustering] WITH FILTER(Region='Europe')

FROM [TM Clustering]
```

Note

Filters that are applied to the case table can be altered by using the SELECT INTO statement as shown in this example; however, if the original model contains a filter on a nested table, the nested table filter cannot be altered or removed by using this syntax, but is copied unchanged from the original model. To create a model with a different filter on a nested table, use the ALTER STRTUCTURE...ADD MODEL syntax.

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Manipulation Statements

The following table lists the statements that are part of the data mining data manipulation language (DML) in Data Mining Extensions (DMX).

Topic	Description
Data Mining Extensions (DMX) Statement Reference	Clears the trained content from a mining model.
INSERT INTO (DMX)	Trains a mining model.
SELECT (DMX)	Browses a mining model.
<source data query>	Queries data sources for both **INSERT INTO** and **SELECT** statements.
UPDATE (DMX)	Changes the content in the mining model.

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Statement Reference

DELETE

Clears a mining model, a mining structure, or a mining structure and all its associated mining models, depending on the Data Mining Extensions (DMX) clauses that you use.

Syntax

```
DELETE FROM [MINING MODEL] <model>[.CONTENT]
DELETE FROM [MINING STRUCTURE] <structure>[.CONTENT]|[.CASES]
```

Arguments

model

A model identifier.

structure

A structure identifier.

Remarks

If you do not specify **MINING MODEL** or **MINING STRUCTURE**, Analysis Services searches for the object type based on the name, and processes the correct object. If the server contains a mining structure and a mining model that have the same name, an error is returned.

The following table describes the result of using different forms of the syntax.

Statement	Result

DELETE FROM MINING STRUCTURE *<structure>*
or
DELETE FROM MINING STRUCTURE *<structure>***.CONTENT**
	Performs **ProcessClear** on the mining structure. All the content is cleared from the mining structure and its associated mining models.

DELETE FROM MINING STRUCTURE *<structure>***.CASES**	Performs **ProcessClearStructureOnly** on the mining structure. All the content is cleared from the mining structure, leaving its associated mining models intact. Drillthrough on the associated mining models will fail after the mining structure has been cleared.

DELETE FROM MINING MODEL *<model>*
or
DELETE FROM MINING MODEL *<model>***.CONTENT**
	Performs **ProcessClear** on the mining model but leaves the state values intact. State values are the possible states of a column. For example, state values for a gender column would be male and female.

For more information about processing types, see Data Mining Extensions (DMX) Statement Reference.

Examples

The following example removes all of the content from the NB_Sample model.

```
DELETE FROM NB_Sample.CONTENT
```

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

INSERT INTO

Processes the specified data mining object. For more information about processing mining models and mining structures, see Data Mining Extensions (DMX) Statement Reference.

If a mining structure is specified, the statement processes the mining structure and all its associated mining models. If a mining model is specified, the statement processes just the mining model.

Syntax

```
INSERT INTO [MINING MODEL]|[MINING STRUCTURE] <model>|<structure> (<mapped model columns>) <source data query>
INSERT INTO [MINING MODEL]|[MINING STRUCTURE] <model>|<structure>.COLUMN_VALUES (<mapped model columns>) <source data query>
```

Arguments

model

A model identifier.

structure

A structure identifier.

mapped model columns

A comma-separated list of column identifiers and nested identifiers.

source data query

The source query in the provider-defined format.

Remarks

If you do not specify **MINING MODEL** or **MINING STRUCTURE**, Analysis Services searches for the object type based on the name, and processes the correct object. If the server contains a mining structure and a mining model that have the same name, an error is returned.

By using the second syntax form, **INSERT INTO** <*object*>**.COLUMN_VALUES**, you can insert data directly into the model columns without training the model. This method provides column data to the model in a concise, ordered manner that is useful when you work with datasets that contain hierarchies or ordered columns.

If you use **INSERT INTO** with a mining model or a mining structure, and leave off the <mapped model columns> and <source data query> arguments, the statement behaves like **ProcessDefault**, using bindings that already exist. If bindings do not exist, the statement returns an error. For more information about **ProcessDefault**, see Processing Options. The following example shows the syntax:

```
INSERT INTO [MINING MODEL] <model>
```

If you specify **MINING MODEL** and provide mapped columns and a source data query, the model and associated structure is processed.

The following table provides a description of the result of different forms of the statement, depending on the state of the objects.

Statement	State of objects	Result
INSERT INTO MINING MODEL <*model*>	Mining structure is processed.	Mining model is processed.
	Mining structure is unprocessed.	Mining model and mining structure are processed.
	Mining structure contains additional mining models.	Process fails. You must reprocess the structure, and the associated mining models.
INSERT INTO MINING STRUCTURE <*structure*>	Mining structure is processed or unprocessed.	Mining structure and associated mining models are processed.
INSERT INTO MINING MODEL <*model*> that contains a source query or **INSERT INTO MINING STRUCTURE** <*structure*> that contains a source query	Either the structure or the model already contains content.	Process fails. You must clear the objects before you perform this operation, by using DELETE (DMX).

Mapped Model Columns

By using the <mapped model columns> element, you can map the columns from the data source to the columns in your mining model. The <mapped model columns> element has the following form:

```
<column identifier> | SKIP | <table identifier> (<column identifier> |
SKIP), ...
```

By using **SKIP**, you can exclude certain columns that must exist in the source query, but that do not exist in the mining model. SKIP is useful when you do not have control over the columns that are included in the input rowset. If you are writing your own OPENQUERY, the better practice is to omit the column from the SELECT column list instead of using SKIP.

SKIP is also useful when a column from the input rowset is needed to perform a join, but the column is not used by the mining structure. A typical example of this is a mining structure and mining model that contain a nested table. The input rowset for this structure will have a foreign key column that is used to create a hierarchical rowset using the SHAPE clause, but the foreign key column is almost never used in the model.

The syntax for SKIP requires that you insert SKIP at the position of the individual column in the input rowset that has no corresponding mining structure column. For example, in the nested table example below, OrderNumber must be selected in the APPEND clause so that it can be used in the RELATE clause to specify the join; however, you do not want to insert the OrderNumber data into the nested table in the mining structure. Therefore, the example uses the SKIP keyword instead of OrderNumber in the INSERT INTO argument.

Source Data Query

The <source data query> element can include the following data source types:

- **OPENQUERY**
- **OPENROWSET**
- **SHAPE**
- Any Analysis Services query that returns a rowset

For more information about data source types, see <source data query>.

Basic Example

The following example uses **OPENQUERY** to train a Naive Bayes model based on the targeted mailing data in the database.

```
INSERT INTO NBSample (CustomerKey, Gender, [Number Cars Owned],

    [Bike Buyer])

OPENQUERY([AdventureWorksDW2012],'Select CustomerKey, Gender,
[NumberCarsOwned], [BikeBuyer]

FROM [vTargetMail]')
```

Nested Table Example

The following example uses **SHAPE** to train an association mining model that contains a nested table. Note that the fist line contains **SKIP** instead OrderNumber, which is required in the **SHAPE_APPEND** statement but is not used in the mining model.

```
INSERT INTO MyAssociationModel

    ([OrderNumber],[Models] (SKIP, [Model])
    )
SHAPE {
    OPENQUERY([AdventureWorksDW2012],'SELECT OrderNumber
    FROM vAssocSeqOrders ORDER BY OrderNumber')
} APPEND (
    {OPENQUERY([AdventureWorksDW2012],'SELECT OrderNumber, model FROM
    dbo.vAssocSeqLineItems ORDER BY OrderNumber, Model')}
  RELATE OrderNumber to OrderNumber)
AS [Models]
```

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

SELECT

The **SELECT** statement in Data Mining Extensions (DMX) is used for the following tasks in data mining:

- Browsing the content of an existing mining model
- Creating predictions from an existing mining model
- Creating a copy of an existing mining model
- Browsing the mining structure

Although the full syntax of this statement is complex, the primary clauses used for browsing a model and its underlying structure can be summarized as follows:

```
SELECT [FLATTENED] [TOP <n>] <select list>
FROM <model/structure>[.aspect]
[WHERE <condition expression>]
[ORDER BY <expression>[DESC|ASC]]
```

FLATTENED

Some data mining clients cannot accept result sets in hierarchical format from a data mining provider. The client may lack the ability to handle a hierarchy, or it may have to store the results in a single denormalized table. To convert the data from nested tables to flattened tables, you must request that the query results be flattened.

To flatten the query results, use the **SELECT** syntax with the **FLATTENED** option, as shown in the following example:

```
SELECT FLATTENED <select list> FROM ...
```

TOP <n> and ORDER BY

You can order the results of a query by using an expression, and can then return a subset of the results by using a combination of the **ORDER BY** and **TOP** clauses. This is useful in a scenario such as targeted mailing where you only want to send results to the most likely respondents. You could order the results of a target mailing prediction query by the prediction probability, and then only return the top <n> results.

Select List

The <select list> can include scalar column references, prediction functions, and expressions. The options that are available depend on the algorithm, and the following contexts:

- Whether you are querying a mining structure or a mining model
- Whether you are querying content or cases
- Whether source data is a relational table or a cube
- If you are making predictions

In many cases, you can use aliases, or create simple expressions based on the items in the select list. For example, the following example shows a simple expression on model columns:

```
SELECT [CustomerID], [Last Name] + ', ' + [FirstName] AS FullName

FROM <model>.CASES
```

The following example creates an alias for a column that contains the results of a prediction function:

```
SELECT Predict([Column1], 'Value') as Column1Prediction

FROM MyModel

JOIN <source data query>
```

WHERE

You can limit the cases that are returned by the query by using a **WHERE** clause. The **WHERE** clause specifies that column references in the **WHERE** expression must have the same semantics as column references in the <select list> of the **SELECT** statement, and can only return a Boolean expression. The syntax for the **WHERE** clause is as follows

```
WHERE < condition expression >
```

The select list and **WHERE** clause of a **SELECT** statement must follow the following rules:

- The select list must contain an expression that does not return a Boolean result. You can modify the expression, but the expression must return non-Boolean results.
- The **WHERE** clause must contain an expression that returns a Boolean result. You can modify the clause, but it must return a Boolean result.

Predictions

There are two types of syntax that you can use for creating predictions:

- SELECT FROM PREDICTION JOIN
- SELECT FROM <model>

The first type of prediction enables you create complex predictions either in real time or as a batch.

The second prediction type creates an empty prediction join on a predictable column in a mining model, and returns the most likely state of the column. The results of this query are completely based on the content of the mining model.

You can insert a select statement into the source query of a **SELECT FROM PREDICTION JOIN** statement by using the following syntax.

```
SELECT FROM PREDICTION JOIN (<SELECT statement>) AS t, WHERE <SELECT statement>
```

For more information about creating prediction queries, see Prediction Queries.

Clause Syntax

Because of the complexity of browsing with the **SELECT** statement, detailed syntax elements and arguments are described by clause. For more information about each clause, click a topic in the following list:

SELECT DISTINCT FROM <model> (DMX)

SELECT FROM <model>.CONTENT (DMX)

SELECT FROM <model>.CASES (DMX)

SELECT FROM <model>.SAMPLE_CASES (DMX)

SELECT FROM <model>.DIMENSION_CONTENT (DMX)

SELECT FROM PREDICTION JOIN (DMX)

SELECT FROM <model> (DMX)

SELECT FROM <structure>.CASES

See Also

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

DMX Data Manipulation Statements

SELECT DISTINCT FROM <model >

Returns all possible states for the selected column in the model. The values that are returned vary depending on whether the specified column contains discrete values, discretized numeric values, or continuous numeric values.

Syntax

SELECT [FLATTENED] DISTINCT [TOP <**n**>] <**expression list**> FROM <**model**>

[WHERE <**`condition list`**>][ORDER BY <**`expression`**>]

Arguments

n

Optional. An integer specifying how many rows to return.

expression list

A comma-separated list of related column identifiers (derived from the model) or expressions.

model

A model identifier.

condition list

A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

The **SELECT DISTINCT FROM** statement only works with a single column or with a set of related columns. This clause does not work with a set of unrelated columns.

The **SELECT DISTINCT FROM** statement allows you to directly reference a column inside of a nested table. For example:

```
<model>.<table column reference>.<column reference>
```

The results of the **SELECT DISTINCT FROM <model>** statement vary, depending on the column type. The following table describes the supported column types and the output from the statement.

Column type	Output
Discrete	The unique values in the column.
Discretized	The midpoint for each discretized bucket in the column.
Continuous	The midpoint for the values in the column.

Discrete Column Example

The following code sample is based on the `[TM Decision Tree]` model that you create in the Basic Data Mining Tutorial. The query returns the unique values that exist in the discrete column, `Gender`.

```
SELECT DISTINCT [Gender]

FROM [TM Decision Tree]
```

Example results:

Gender
F
M

For columns that contain discrete values, the results always include the Missing state, shown as a null value.

Continuous Column Example

The following code sample returns the midpoint, minimum age, and maximum age for all of the values in the column.

```
SELECT DISTINCT [Age] AS [Midpoint Age],
    RangeMin([Age]) AS [Minimum Age],
    RangeMax([Age]) AS [Maximum Age]
FROM [TM Decision Tree]
```

Example results:

Midpoint Age	Minimum Age	Maximum Age
62	26	97

The query also returns a single row of null values, to represent missing values.

Discretized Column Example

The following code sample returns the midpoint, maximum, and minimum values for each bucket that has been created by the algorithm for the column, `[Yearly Income]`. To reproduce the results for this example, you must create a new mining structure that is the same as `[Targeted Mailing]`. In the wizard, change the content type of the `Yearly Income` column from **Continuous** to **Discretized**.

Note

You can also change the mining model created in the Basic Mining Tutorial to discretize the mining structure column, `[Yearly Income]`. For information about how to do this, see How to: Change the Discretization of a Column in a Mining Model. However, when you change the discretization of the column, it will force the mining structure to be reprocessed, which will change the results of other models that you have built using that structure.

```
SELECT DISTINCT [Yearly Income] AS [Bucket Average],
    RangeMin([Yearly Income]) AS [Bucket Minimum],
    RangeMax([Yearly Income]) AS [Bucket Maximum]
FROM [TM Decision Tree]
```

Example results:

Bucket Average	Bucket Minimum	Bucket Maximum
24610.7	10000	39221.41
55115.73	39221.41	71010.05
84821.54	71010.05	98633.04
111633.9	98633.04	124634.7
147317.4	124634.7	170000

You can see that the values of the [Yearly Income] column have been discretized into five buckets, plus an additional row of null values, to represent missing values.

The number of decimal places in the results depends on the client that you use for querying. Here they have been rounded to two decimal places, both for simplicity and to reflect the values that are displayed in SQL Server Data Tools (SSDT).

For example, if you browse the model by using the Decision Tree viewer and click a node that contains customers grouped by income, the following node properties are displayed in the Tooltip:

Age >=69 AND Yearly Income < 39221.41

Note

The minimum value of the minimum bucket and the maximum value of the maximum bucket are just the highest and lowest observed values. Any values that fall outside this observed range are assumed to belong to the minimum and maximum buckets.

See Also

Data Mining Extensions (DMX) Statement Reference

DMX Data Manipulation Statements

DMX Statement Reference

SELECT FROM <model>.CONTENT

Returns the mining model schema rowset for the specified data mining model.

Syntax

```
SELECT [FLATTENED] [TOP <n>] <expression list> FROM <model>.CONTENT
[WHERE <condition expression>]
[ORDER BY <expression> [DESC|ASC]]
```

Arguments

n

Optional. An integer that specifies how many rows to return.

expression list

A comma-separated list of columns derived from the Content schema rowset.

model

A model identifier.

condition expression

Optional. A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

The **SELECT FROM** <*model*>**.CONTENT** statement returns content that is specific to each algorithm. For example, you might want to use the descriptions of all the rules of an association rules model in a custom application. You can use a **SELECT FROM <model>.CONTENT** statement to return values in the NODE_RULE column of the model.

The following table lists the columns that are included in the mining model content.

Note
Algorithms might interpret the columns differently in order to correctly represent the content. For a description of the mining model content for each algorithm, and tips on how to interpret and query the mining model content for each model type, see Mining Model Content (Analysis Services - Data Mining).

CONTENT rowset column	Description
MODEL_CATALOG	A catalog name. NULL if the provider does not support catalogs.

CONTENT rowset column	**Description**
MODEL_SCHEMA	An unqualified schema name. NULL if the provider does not support schemas.
MODEL_NAME	A model name. This column cannot contain a NULL.
ATTRIBUTE_NAME	The name of the attribute that corresponds to the node.
NODE_NAME	The name of the node.
NODE_UNIQUE_NAME	The unique name of the node within the model.
NODE_TYPE	An integer that represents the type of the node. .
NODE_GUID	The node GUID. NULL if no GUID.
NODE_CAPTION	A label or a caption that is associated with the node. Used primarily for display purposes. If a caption does not exist, NODE_NAME is returned.
CHILDREN_CARDINALITY	The number of children that the node has.
PARENT_UNIQUE_NAME	The unique name of the node's parent.
NODE_DESCRIPTION	A description of the node.
NODE_RULE	An XML fragment that represents the rule embedded in the node. The format of the XML string is based on the PMML standard.
MARGINAL_RULE	An XML fragment that describes the path from the parent to the node.
NODE_PROBABILITY	The probability of the path that ends in the node.
MARGINAL_PROBABILITY	The probability of reaching the node from the parent node.
NODE_DISTRIBUTION	A table that contains statistics that describe the distribution of values in the node.
NODE_SUPPORT	The number of cases in support of this node.

Examples

The following code returns the ID of the parent node for the decision trees model that was added to the Targeted Mailing mining structure.

```
SELECT MODEL_NAME, NODE_NAME FROM [TM Decision Tree].CONTENT

WHERE NODE_TYPE = 1
```

Expected results:

MODEL_NAME	NODE_NAME
TM_DecisionTree	0

The following query uses the **IsDescendant** function to return the immediate children of the node that was returned in the previous query.

Note
Because the value of the NODE_NAME is a string, you cannot use a sub-select statement to return the NODE_ID as an argument to the **IsDescendant** function.

```
SELECT NODE_NAME, NODETYPE, NODE_CAPTION

FROM [TM Decision Tree].CONTENT

WHERE ISDESCENDANT('0')
```

Expected results:

Because the model is a decision trees model, the descendants of the model parent node include a single marginal statistics node, a node that represents the predictable attribute, and multiple nodes that contain input attributes and values. For more information, see Mining Model Content for Decision Tree Models (Analysis Services - Data Mining).

Using the FLATTENED Keyword

The mining model content frequently contains interesting information about the model in nested table columns. The FLATTENED keyword lets you retrieve data from a nested table column without using a provider that supports hierarchical rowsets.

The following query returns a single node, the marginal statistics node (NODE_TYPE = 26) from a Naïve Bayes model. However, this node contains a nested table, in the NODE_DISTRIBUTION column. As a result, the nested table column is flattened and a row is returned for every row in the nested table. The value of the scalar column MODEL_NAME is repeated for each row in the nested table.

Also, notice that if you specify only the name of the nested table column, a new column is returned for each column in the nested table. By default, the name of the nested table is prefixed to the name of each nested table column.

```
SELECT FLATTENED MODEL_NAME, NODE_DISTRIBUTION
```

```
FROM [TM_NaiveBayes].CONTENT
WHERE NODE_TYPE = 26
```

Example results:

MODEL_NAME	NODE_DISTRIBUTION.ATTRIBUTE_NAME	NODE_DISTRIBUTION.ATTRIBUTE_VALUE	NODE_DISTRIBUTION.SUPPORT	NODE_DISTRIBUTION.PROBABILITY	NODE_DISTRIBUTION.VARIANCE	NODE_DISTRIBUTION.VALUETYPE
TM_NaiveBayes	Bike Buyer	Missing	0	0	0	1
TM_NaiveBayes	Bike Buyer	0	6556	0.506685215240745	0	
TM_NaiveBayes	Bike Buyer	1	6383	0.493314784759255	0	

The following example demonstrates how to return only some of the columns from the nested table by using a sub-select statement. You can simplify the display by aliasing the table name of the nested table, as shown.

```
SELECT MODEL_NAME,
(SELECT ATTRIBUTE_NAME, ATTRIBUTE_VALUE, [SUPPORT] AS t
FROM NODE_DISTRIBUTION)
FROM TM_NaiveBayes.CONTENT
WHERE NODE_TYPE = 26
```

Example results:

MODEL_NAME	t.ATTRIBUTE_NAME	t.ATTRIBUTE_VALUE	t.SUPPORT
TM_NaiveBayes	Bike Buyer	Missing	0
TM_NaiveBayes	Bike Buyer	0	6556
TM_NaiveBayes	Bike Buyer	1	6383

See Also

SELECT (DMX)

DMX Data Manipulation Statements

DMX Statement Reference

SELECT FROM <model>.CASES

Supports drillthrough, and returns the cases that were used to train the model. You can also return structure columns that are not included in the model, if drillthrough has been enabled on the mining structure and on the mining model, and if you have the appropriate permissions.

If drillthrough is not enabled on the mining model, this statement will fail.

Note

In Data Mining Extensions (DMX) you can only enable drillthrough when you create the model. You can add drillthrough to an existing model by using SQL Server Data Tools (SSDT), but the model must be reprocessed before you can view or query the cases.

For more information about how to enable drillthrough, see Data Mining Extensions (DMX) Statement Reference, SELECT INTO (DMX), and ALTER MINING STRUCTURE (DMX).

Syntax

SELECT [FLATTENED] [TOP <n>] <**expression list**> FROM <**model**>.CASES

[WHERE <**condition expression**>][ORDER BY <**expression**> [DESC|ASC]]

Arguments

n

Optional. An integer that specifies how many rows to return.

expression list

A comma-separated list of expressions. An expression can include column identifiers, user-defined functions, UDFs, and VBA functions, and others.

To include a structure column that is not included in the mining model, use the function `StructureColumn('<structure column name>')`.

model

A model identifier.

condition expression

A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

If drillthrough is enabled on both the mining model and the mining structure, users who are member of a role that has drillthrough permission on the model and the structure can access columns of the mining structure that are not included in the mining model. Therefore, to protect sensitive data or personal information, you should construct your data source view to mask personal information, and grant **AllowDrillthrough** permission on a mining structure only when it is necessary.

The Lag function can be used with time series models to return or filter on the time lag between each case and the initial time.

Using the IsInNode function in the **WHERE** clause returns only cases that are associated with the node that is specified by the NODE_UNIQUE_NAME column of the schema rowset.

Examples

The following examples are based on the mining structure Targeted Mailing, which is based on the database and its associated mining models. For more information, see Basic Data Mining Tutorial.

Example 1: Drillthrough to Model Cases and Structure Columns

The following example returns the columns for all the cases that were used to test the Targeted Mailing model. If the mining structure on which the model is built does not have a holdout test data set, this query would return 0 cases. You can use the expression list to return only the columns that you need.

```
SELECT * FROM [TM Decision Tree].Cases

WHERE IsTestCase();
```

Example 2: Drillthrough to Training Cases in a Specific Node

The following example returns just those cases that were used to train Cluster 2. The node for Cluster 2 has the value '002' for the NODE_UNIQUE_NAME column. The example also returns one structure column, [Customer Key], that was not a part of the mining model, and provides the alias `CustomerID` for the column. Note that the name of the structure column is passed as a string value and therefore must be enclosed in quotation marks, not brackets.

```
SELECT StructureColumn('Customer Key') AS CustomerID, *

FROM [TM_Clustering].Cases

WHERE IsTrainingCase()

AND IsInNode('002')
```

To return a structure column, drillthrough permissions must be enabled on both the mining model and the mining structure.

Note

Not all mining model types support drillthrough. For information about the models that support drillthrough, see Using Drillthrough on Mining Models and Mining Structures (Analysis Services - Data Mining).

See Also

SELECT (DMX)

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

SELECT FROM <model>.SAMPLE_CASES

Returns sample cases that are representative of the cases that are used to train the data mining model.

To use this statement, you must enable drillthrough when you create the mining model. For more information about enabling drillthrough, see Data Mining Extensions (DMX) Statement Reference, SELECT INTO (DMX), and ALTER MINING STRUCTURE (DMX).

Syntax

```
SELECT [FLATTENED] [TOP <n>] <expression list> FROM <model>.SAMPLE_CASES
[WHERE <condition list>] ORDER BY <expression> [DESC|ASC]]
```

Arguments

n

Optional. An integer that specifies how many rows to return.

expression list

A comma-separated list of related column identifiers.

model

A model identifier.

condition list

Optional. Conditions to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

Sample cases may be generated and may not actually exist in the training data. The returned case is representative of the specified content node.

Although the Microsoft Sequence Clustering algorithm is the only Microsoft algorithm that supports using **SELECT FROM <model>.SAMPLE_CASES**, third-party algorithms may also support it.

Examples

The following example returns sample cases that are used to train the Target Mail mining model. Using the IsInNode function in the **WHERE** clause returns only cases that are associated with the '000000003' node. The node string can be found in the NODE_UNIQUE_NAME column of the schema rowset.

```
Select * from [Sequence Clustering].SAMPLE_Cases

WHERE IsInNode('000000003')
```

See Also

SELECT (DMX)

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

SELECT FROM <model>.DIMENSION_CONTENT

A mining model can be used as a dimension in an OLAP cube, where each node in the model is represented as a member of the dimension. **The SELECT FROM <model>.Dimension_CONTENT** statement returns the content of the model that pertains to its usage as a dimension.

Syntax

```
SELECT [FLATTENED] [TOP <n>] <expression list> FROM <model>.Dimension_CONTENT
[WHERE <condition expression>]
[ORDER BY <expression> [DESC|ASC]]
```

Arguments

n

Optional. An integer that specifies how many rows to return.

expression list

A comma-separated list of related column identifiers derived from the content schema rowset.

model

A model identifier.

condition expression

Optional. A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

Algorithm providers define which content is returned, and how to organize it. For example, the provider might limit the number of nodes that are described in the dimension content.

The following table lists the columns that can be queried for the dimension content, and the function that each column performs as a data mining dimension.

CONTENT rowset column	Function in data mining dimension
ATTRIBUTE_NAME	Member property.
NODE_NAME	Member property.
NODE_UNIQUE_NAME	Key attribute.
NODE_TYPE	Member property.
NODE_CAPTION	CaptionColumn for **Key** attribute.
CHILDREN_CARDINALITY	Member property.
PARENT_UNIQUE_NAME	RelatedAttribute for **Key** attribute (ParentAttribute in parent-child hierarchy).
NODE_DESCRIPTION	Member property.
NODE_RULE	Member property.
MARGINAL_RULE	Member property.
NODE_PROBABILITY	Member property.
MARGINAL_PROBABILITY	Member property.
NODE_SUPPORT	Member property.

Examples

Description

The example selects all columns from the `[TM Decision Tree]` model content that pertain to using the model as a dimension.

Code

```
SELECT *

FROM [TM Decision Tree].Dimension_Content
```

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Definition Statements

[DMX Data Manipulation Statements](#)

[DMX Statement Reference](#)

SELECT FROM <model> PREDICTION JOIN

Uses a mining model to predict the states of columns in an external data source. The **PREDICTION JOIN** statement matches each case from the source query to the model.

Syntax

```
SELECT [FLATTENED] [TOP <n>] <select expression list>
FROM <model> | <sub select> [NATURAL] PREDICTION JOIN
<source data query> [ON <join mapping list>]
[WHERE <condition expression>]
[ORDER BY <expression> [DESC|ASC]]
```

Arguments

n

Optional. An integer that specifies how many rows to return.

select expression list

A comma-separated list of column identifiers and expressions that are derived from the mining model.

model

A model identifier.

sub select

An embedded select statement.

source data query

The source query.

join mapping list

Optional. A logical expression that compares columns from the model to columns from the source query.

condition expression

Optional. A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

The ON clause defines the mapping between the columns from the source query and the columns from the mining model. This mapping is used to direct columns from the source query to columns in the mining model so that the columns can be used as inputs to create the predictions. Columns in the <join mapping list> are related by using an equal sign (=), as shown in the following example:

```
[MiningModel].ColumnA = [source data query].Column1 AND

[MiningModel].ColumnB = [source data query].Column2 AND

...
```

If you are binding a nested table in the ON clause, ensure that you bind the key column with any non-key columns so that the algorithm can correctly identify which case the record of the nested column belongs to.

The source query for the prediction join can either be a table or a singleton query.

You can specify prediction functions that do not return a table expression in the <select expression list> and the <condition expression>.

NATURAL PREDICTION JOIN automatically maps together column names from the source query that match column names in the model. If you use **NATURAL PREDICTION**, you can omit the ON clause.

The WHERE condition can be applied only to predictable columns or related columns.

The ORDER by clause can accept only a single column as an argument; that is, you cannot sort on more than one column.

Example 1: Singleton Query

The following example shows how to create a query to predict whether a specific person will buy a bicycle in real time. In this query the data is not stored in a table or other data source, but instead is entered directly into the query. The person in the query has the following traits:

- 35 years old
- Owns a house
- Owns two cars
- Has two children living at home

Using the TM Decision Tree mining model and the known characteristics about the subject, the query returns a Boolean value that describes whether the person bought the bike and a set of tabular values, returned by the PredictHistogram (DMX) function, that describe how the prediction was made.

```
SELECT

  [TM Decision Tree].[Bike Buyer],

  PredictHistogram([Bike Buyer])

FROM

  [TM Decision Tree]

NATURAL PREDICTION JOIN
```

```
(SELECT 35 AS [Age],
  '5-10 Miles' AS [Commute Distance],
  '1' AS [House Owner Flag],
  2 AS [Number Cars Owned],
  2 AS [Total Children]) AS t
```

Example 2: Using OPENQUERY

The following example shows how to create a batch prediction query by using a list of potential customers stored in an external dataset. Because the table is part of a data source view that has been defined on an instance of Analysis Services, the query can use OPENQUERY to retrieve the data. Because the names of the columns in the table are different from those in the mining model, the **ON** clause must be used to map the columns in the table to the columns in the model.

The query returns the first and last name of each person in the table, together with a Boolean column that indicates whether each person is likely to buy a bike, where 0 means "probably will not buy a bike" and 1 means "probably will buy a bike". The last column contains the probability for the predicted result.

```
SELECT
  t.[LastName],
  t.[FirstName],
  [TM Decision Tree].[Bike Buyer],
  PredictProbability([Bike Buyer])
From
  [TM Decision Tree]
PREDICTION JOIN
  OPENQUERY([Adventure Works DW Multidimensional 2012],
    'SELECT
      [LastName],
      [FirstName],
      [MaritalStatus],
      [Gender],
      [YearlyIncome],
      [TotalChildren],
      [NumberChildrenAtHome],
      [Education],
      [Occupation],
      [HouseOwnerFlag],
      [NumberCarsOwned]
    FROM
      [dbo].[ProspectiveBuyer]
    ') AS t
ON
  [TM Decision Tree].[Marital Status] = t.[MaritalStatus] AND
  [TM Decision Tree].[Gender] = t.[Gender] AND
  [TM Decision Tree].[Yearly Income] = t.[YearlyIncome] AND
  [TM Decision Tree].[Total Children] = t.[TotalChildren] AND
  [TM Decision Tree].[Number Children At Home] =
t.[NumberChildrenAtHome] AND
  [TM Decision Tree].[Education] = t.[Education] AND
  [TM Decision Tree].[Occupation] = t.[Occupation] AND
  [TM Decision Tree].[House Owner Flag] = t.[HouseOwnerFlag] AND
  [TM Decision Tree].[Number Cars Owned] = t.[NumberCarsOwned]
```

To restrict the data set to only the customers who are predicted to buy a bike, and then sort the list by customer name, you can add a WHERE clause and an ORDER BY clause to the previous example:

```
WHERE [BIKE Buyer]
ORDER BY [LastName] ASC
```

Example 3: Predicting Associations

The following example shows how to create a prediction by using a model that is built from the Microsoft Association algorithm. Predictions on an association model can be used to recommend related products. For example, the following query returns the three products that are most likely to be purchased together:

- Mountain Bottle Cage
- Mountain Tire Tube
- Mountain-200

The Predict (DMX) function is polymorphic and can be used with all model types. You use the value3 as an argument to the function to limit the number of items that are returned by the query. The **SELECT** list that follows the NATURAL PREDICTION JOIN clause supplies the values to use as input for prediction.

```
SELECT FLATTENED
  PREDICT([Association].[v Assoc Seq Line Items], 3)
FROM
```

```
  [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Mountain Bottle Cage' AS [Model]
  UNION SELECT 'Mountain Tire Tube' AS [Model]
  UNION SELECT 'Mountain-200' AS [Model]) AS [v Assoc Seq Line Items ])
AS t
```

Example results:

Expression.Model
HL Mountain Tire
Water Bottle
Fender Set - Mountain

Because the column that contains the predictable attribute, `[v Assoc Seq Line Items]`, is a table column, the query returns a single column that contains a nested table. By default the nested table column is named `Expression`. If your provider does not support hierarchical rowsets, you can use the **FLATTENED** keyword as shown in this example to make the results easier to view.

See Also

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Data Definition Statements

DMX Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

SELECT FROM <model>

Performs an empty prediction join, returning the most probable value or values for the specified columns. Only the content from the mining model is used to create the prediction.

Syntax

SELECT <**`expression list`**> [TOP <**`n`**>] FROM <**`model`**>

[WHERE <**`condition list`**>]

[ORDER BY <**`expression`**> [DESC|ASC]]

Arguments

expression list

A comma-separated list of expressions, or of predict or predict only columns.

n

Optional. An integer that specifies how many rows to return.

model

A model identifier.

condition list

Optional. Conditions to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

The columns in the expression list must be defined as predict or predict only, or related to a predictable column.

Naive Bayes Example

The following example performs an empty prediction join on the Bike Buyer column, returning the most likely state in the TM Naive Bayes mining model.

```
SELECT ([Bike Buyer]) FROM [TM_Naive_Bayes]
```

Time Series Example

The following example performs a prediction on the Amount column in the Forecasting model, returning the next four time steps. The Model Region column combines bike models and regions into a single identifier. The query uses the Data Mining Extensions (DMX) Statement Reference function to perform the prediction.

```
SELECT [Model Region], PredictTimeSeries(Amount, 4)

FROM Forecasting
```

See Also

SELECT (DMX)

Data Mining Extensions (DMX) Data Definition Statements

DMX Data Manipulation Statements

DMX Statement Reference

SELECT FROM <structure>.CASES

Returns the cases that were used to create the mining structure.

If drillthrough is not enabled on the structure, the statement will fail. Also, the statement will fail if the user does not have drillthrough permissions on the mining structure.

In Analysis Services, drillthrough on new mining structures is enabled by default. To verify whether drillthrough is enabled for a particular structure, check whether the value of the **CacheMode** property is set to **KeepTrainingCases**.

If the value of **CacheMode** is changed to **ClearAfterProcessing**, the structure cases are cleared from the cache and you cannot use drillthrough.

Note
You cannot enable or disable drillthrough on the mining structure by using Data Mining Extensions (DMX).

Syntax

SELECT [TOP n] <expression `list`> FROM <`structure`>.CASES

[WHERE <`condition expression`>][ORDER BY <`expression`> [DESC|ASC]]

Arguments

n

Optional. An integer that specifies how many rows to return.

expression list

A comma-separated list of expressions.

An expression can include column identifiers, user-defined functions, and VBA functions.

structure

The name of the structure.

condition expression

A condition to restrict the values that are returned from the column list.

expression

Optional. An expression that returns a scalar value.

Remarks

If drillthrough is enabled on both the model and the structure, any member of a role that has drillthrough permissions on the mining structure and the model can return structure columns that were not included in the model, by using the following syntax:

```
SELECT StructureColumn('<column name>') FROM <model>.CASES
```

Therefore, to protect sensitive data or personal information, you should construct your data source view to mask personal information, and grant **AllowDrillthrough** permission on a mining structure or mining model only when necessary.

Examples

The following examples are based on the mining structure, Targeted Mailing, which is based on the Adventure Works DW Multidimensional 2012 database, and the associated mining models. For more information, see Basic Data Mining Tutorial.

Example 1: Drill through to Structure Cases

The following example returns a list of the 500 oldest customers in the mining structure, Targeted Mailing. The query returns all the columns in the mining model, but restricts the rows to those who purchased a bike, and orders them by age. You can also edit the expression list to return only the columns that you need.

```
SELECT TOP 500 *

FROM [Targeted Mailing].Cases

WHERE [Bike Buyer] = 1

ORDER BY Age DESC;
```

Example 2: Drillthrough to Test or Training Cases Only

The following example returns a list of the structure cases for Targeted Mailing that are reserved for testing. If the mining structure does not contain a holdout test set, by default all cases are treated as training cases, and this query would return 0 cases.

```
SELECT [Customer Key], Gender, Age

FROM [Targeted Mailing].Cases

WHERE IsTestCase();
```

To return the training cases, substitute the function `IsTrainingCase()`.

See Also

SELECT (DMX)

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Data Mining Extensions (DMX) Statement Reference

<source data query>

To train a data mining model and create predictions from a mining model, you have to access data that is external to the Microsoft SQL Server Analysis Services database. You use the <source data query> clause in Data Mining Extensions (DMX) to define this external data. The INSERT INTO (DMX), Nested Tables, and SELECT FROM NATURAL PREDICTION JOIN statements all use **<source data query>**.

Query types

The three most common ways to specify source data are:

OPENQUERY

This statement queries data that is external to an instance of Analysis Services, by using an existing data source.

While **OPENQUERY** is similar in function to **OPENROWSET**, **OPENQUERY** has the following benefits:

- A DMX query is much easier to write with **OPENQUERY**. Instead of creating a new connection string every time that you write a query, you can take advantage of the existing connection string in the data source. The data source object can also control data access for individual users.

- The administrator has more control over how the data on the server is accessed. For example, the administrator can manage which providers are loaded into the server and which external data can be accessed.

OPENROWSET

This statement queries data that is external to an instance of Analysis Services, by using an existing data source.

SHAPE

This statement queries multiple data sources to create a nested table. By using **SHAPE**, you can combine data from multiple sources into a single hierarchical table. This lets you take advantage of the ability of Analysis Services to nest tables by imbedding a table within a table.

To specify the source data, you can also use the following options:

- Any valid DMX statement
- Any valid Multidimensional Expressions (MDX) statement
- A table that returns a stored procedure
- An XML for Analysis (XMLA) rowset
- A rowset parameter

See Also

DMX Data Manipulation Statements

DMX Statement Reference

Nested Tables

OPENQUERY

Replaces the source data query with a query to an existing data source. The INSERT, SELECT FROM PREDICTION JOIN, and SELECT FROM NATURAL PREDICTION JOIN statements support **OPENQUERY**.

Syntax

```
OPENQUERY(<named datasource>, <query syntax>)
```

Arguments

named datasource

A data source that exists on the Microsoft SQL Server Analysis Services database.

query syntax

A query syntax that returns a rowset.

Remarks

OPENQUERY provides a more secure way to access external data by supporting data source permissions. Because the connection string is stored in the data source, administrators can use the properties of the data source to manage access to the data. For more information about data sources, see Working with Data Sources.

For information about permission issues related to OPENQUERY, see Securing the Data Sources Used by Analysis Services.

You can get a list of the data sources that are available on a server by querying the **MDSCHEMA_INPUT_DATASOURCES** schema rowset. For more information about using **MDSCHEMA_INPUT_DATASOURCES**, see Data Mining Extensions (DMX) Statement Reference.

You can also return a list of data sources in the current Analysis Services database by using the following DMX query:

```
SELECT * FROM $system.MDSCHEMA_INPUT_DATASOURCES
```

Examples

The following example uses the MyDS data source already defined in the Analysis Services database to create a connection to the database and query the **vTargetMail** view.

```
OPENQUERY (MyDS,'SELECT TOP 1000 * FROM vTargetMail')
```

See Also

<source data query>

DMX Data Manipulation Statements

DMX Statement Reference

OPENROWSET

Replaces the source data query with a query to an external provider. The INSERT, SELECT FROM PREDICTION JOIN, and SELECT FROM NATURAL PREDICTION JOIN statements support **OPENROWSET**.

Syntax

OPENROWSET(**provider_name**,**provider_string**,**query_syntax**)

Arguments

provider_name

An OLE DB provider name.

provider_string

The OLE DB connection string for the specified provider.

query_syntax

A query syntax that returns a rowset.

Remarks

The data mining provider will establish a connection to the data source object by using provider_name and provider_string, and will execute the query specified in query_syntax to retrieve the rowset from the source data.

Examples

The following example can be used within a PREDICTION JOIN statement to retrieve data from the   database by using a Transact-SQL SELECT statement.

```
OPENROWSET

(

'SQLOLEDB.1',

'Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=AdventureWorksDW2012;Data Source=localhost',

'SELECT TOP 1000 * FROM vTargetMail'

)
```

See Also

Data Mining Extensions (DMX) Statement Reference

DMX Data Manipulation Statements

DMX Statement Reference

SHAPE

Combines queries from multiple data sources into a single hierarchical table (that is, a table with nested tables), which becomes the case table for the mining model.

The complete syntax of the **SHAPE** command is documented in the Microsoft Data Access Components (MDAC) Software Development Kit (SDK).

Syntax

```
SHAPE {<master query>}
APPEND ({ <child table query> }
   RELATE <master column> TO <child column>)
      AS <column table name>
[
   ({ <child table query> }
    RELATE <master column> TO <child column>)
      AS < column table name>
...
]
```

Arguments

master query

The query returning the parent table.

child table query

The query returning the nested table.

master column

The column in the parent table to identify child rows from the result of a child table query.

child column

The column in the child table to identify the parent row from the result of a master query.

column table name

The newly appended column name in the parent table for the nested table.

Remarks

You must order the queries by the column that relates the parent table and child table.

Examples

You can use the following example within an Data Mining Extensions (DMX) Statement Reference statement to train a model containing a nested table. The two tables within the **SHAPE** statement are related through the **OrderNumber** column.

```
SHAPE {
    OPENQUERY([Adventure Works DW Multidimensional 2012],'SELECT
OrderNumber
    FROM vAssocSeqOrders ORDER BY OrderNumber')
} APPEND (
    {OPENQUERY([Adventure Works DW Multidimensional 2012],'SELECT
OrderNumber, model FROM
    dbo.vAssocSeqLineItems ORDER BY OrderNumber, Model')}
  RELATE OrderNumber to OrderNumber)
```

See Also

<source data query>

Data Mining Extensions (DMX) Data Definition Statements

DMX Data Manipulation Statements

DMX Statement Reference

UPDATE

Changes the **NODE_CAPTION** column in the data mining model.

Syntax

```
UPDATE <model>.CONTENT
SET NODE_CAPTION='new caption'
[WHERE <condition expression>]
```

Arguments

model

A model identifier.

new caption

A string that contains the new name for the **NODE_CAPTION** column.

condition expression

Optional. A condition to restrict the values that are returned from the column list.

Examples

In the following example, the **UPDATE** statement changes the default name, `Cluster 1`, for cluster `001` to the more descriptive name, `Likely Customers`.

```
UPDATE [TM Clustering].CONTENT

SET NODE_CAPTION= 'Likely Customers'

WHERE NODE_UNIQUE_NAME = '001'
```

See Also

Data Mining Extensions (DMX) Statement Reference

DMX Data Manipulation Statements

DMX Statement Reference

Data Mining Extensions (DMX) Function Reference

Analysis Services supports several functions in the Data Mining Extensions (DMX) language. Functions expand the results of a prediction query to include information that further describes the prediction. Functions also provide more control over how the results of the prediction are returned. The following table provides links to resources to help you understand how to use functions in DMX.

Function	**Description**
Mapping Functions to Query Types (DMX)	List functions that can be used with all model types, and provides links to more information about how to query specific types of mining models.
Prediction Queries (DMX)	Provides an overview of how to construct a prediction query by using DMX.
Understanding the Select Statement (DMX)	Returns a table that contains a specified number of bottom-most rows, in increasing order of rank based on a rank expression.

The following table lists the functions that DMX supports.

Function	**Description**
BottomCount	Returns a table that contains the last n-item rows of the table expression, in increasing order based on a rank expression.
BottomPercent	Returns a table that contains the smallest number of bottom-most rows that meet a specified percent expression, in increasing order of rank based on a rank expression.
BottomSum	Returns a table that contains the smallest number of bottom-most rows that meet a specified sum expression, in increasing order of rank based on a rank expression.
Cluster	Returns the cluster that is most likely to contain the input case.
ClusterProbability	Returns the probability that the input case belongs to the cluster.
Exists	Returns true if the result set returned by the specified SELECT statement contains at least one row.

Function	Description
IsDescendant	Indicates whether the current node descends from the specified node.
IsInNode	Indicates whether the specified node contains the case.
IsTestCase	Indicates whether a case belongs to the set of test cases.
IsTrainingCase	Indicates whether a case belongs to the set of training cases.
Lag	Returns the time slice between the date of the current case and the last date in the data.
Predict	Performs a prediction on a specified column.
PredictAdjustedProbability	Returns the adjusted probability of the specified predictable column.
PredictAssociation	Predicts associative membership in a column.
PredictCaseLikelihood	Returns the likelihood that an input case will fit within the existing model. This function can only be used with clustering models.
PredictHistogram	Returns a table that represents the histogram for a specified column.
PredictNodeId	Returns the NodeID for a selected case.
PredictProbability	Returns the probability of the specified column.
PredictSequence	Predicts the next values in a sequence.
PredictStdev	Retrieves the standard deviation value for a specified column.
PredictSupport	Returns the support value of the column.
PredictTimeSeries	Predicts the future values for a time series.
PredictVariance	Returns the variance value of the specified column.

Function	Description
RangeMax	Returns the upper value of the predicted bucket that is discovered for a specified discretized column.
RangeMid	Returns the midpoint value of the predicted bucket that is discovered for a specified discretized column.
RangeMin	Returns the lower value of the predicted bucket that is discovered for a specified discretized column.
StructureColumn	Returns the value of the specified table mining structure column.
TopCount	Returns a table that contains a specified number of topmost rows, in a decreasing order of rank based on a rank expression.
TopPercent	Returns a table that contains the smallest number of topmost rows that meet a specified percent expression, in a decreasing order of rank based on a rank expression.
TopSum	Returns a table that contains the smallest number of topmost rows that meet a specified sum expression, in a decreasing order of rank based on a rank expression.

See Also

Data Mining Extensions (DMX) Operator Reference

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Syntax Conventions

Data Mining Extensions (DMX) Syntax Elements

Mapping Functions to Query Types (DMX)

Prediction Queries (DMX)

Understanding the Select Statement (DMX)

BottomCount

Returns the specified number of bottom-most rows, in increasing order of rank as specified by an expression.

Syntax

BottomCount(<**`table expression`**>, <**`rank expression`**>, <**`count`**>)

Applies To

An expression that returns a table, such as a <table column reference>, or a function that returns a table.

Return Type

<table expression>

Remarks

The value that is supplied by the <rank expression> argument determines the increasing order of rank for the rows that are supplied in the <table expression> argument, and the number of bottom-most rows that is specified in the <count> argument is returned.

Examples

The following example creates a prediction query against the Association model that you build by using the Basic Data Mining Tutorial.

To understand how **BottomCount** works, it might be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)

FROM

     [Association]

NATURAL PREDICTION JOIN

SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note
In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851
Water Bottle	2866	0.192620472	0.175205052
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **BottomCount** function takes the results of this query and returns the smallest-valued rows that sum to the specified percentage.

```
SELECT
BottomCount
    (
    Predict ([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $SUPPORT,
    3)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **BottomCount** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **BottomCount** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $SUPPORT because support values are not fractional and therefore are easier to verify.

The third argument to the **BottomCount** function specifies the number of rows. To get the three lowest-ranked rows, as ordered by $SUPPORT, you type 3.

Example results:

Model	**$SUPPORT**	**$PROBABILITY**	**$ADJUSTEDPROBABILITY**
Road Bottle Cage	1195	0.080314537	0.077173962
Mountain Bottle Cage	1367	0.091874454	0.087780332
Fender Set - Mountain	1415	0.095100477	0.090718432

Note This example is provided only to illustrate the use of **BottomCount**. Depending on the size of your data set, this query might take a long time to run.

See Also

Functions (DMX)

Mapping Functions to Query Types (DMX)

BottomPercent (DMX)

BottomSum (DMX)

TopCount (DMX)

BottomPercent

Returns, in order of increasing rank, the bottom-most rows of a table whose cumulative total is at least a specified percentage.

Syntax

```
BottomPercent(<table expression>, <rank expression>, <percent>)
```

Arguments

<Table expression>

The name of a nested table column or table-valued expression.

<rank expression>

A column in the nested table, or expression that evaluates to a column.

<percent>

A double that indicates the total target percentage.

Result Type

A table.

Remarks

The **BottomPercent** function returns the bottom-most rows in increasing order of rank. The rank is based on the evaluated value of the <rank expression> argument for each row, such that the sum of the <rank expression> values is at least the given percentage that is specified by the <percent> argument. **BottomPercent** returns the smallest number of elements possible while still meeting the specified percent value.

Examples

The following example creates a prediction query against the Association model that you built in the Basic Data Mining Tutorial.

To understand how **BottomPercent** works, it may be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)

FROM

     [Association]

NATURAL PREDICTION JOIN

SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note

In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Water Bottle	2866	0.192620472	0.175205052
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **BottomPercent** function takes the results of this query and returns the smallest-valued rows that sum to the specified percentage.

```
SELECT
BottomPercent
    (
    Predict ([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $SUPPORT,
    50)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **BottomPercent** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **BottomPercent** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $SUPPORT because support values are not fractional and therefore are easier to verify.

The third argument to the **BottomPercent** function specifies the percentage, as a double. To get the rows that represent the bottom 50 percent of the support, you type 50.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Road Bottle Cage	1195	0.080314537	0.077173962
Mountain Bottle Cage	1367	0.091874454	0.087780332
Fender Set - Mountain	1415	0.095100477	0.090718432
Cycling Cap	1473	0.098998589	0.094256014
Road Tire Tube	1588	0.106727603	0.101229538
Mountain-200	1755	0.117951475	0.111260823
Mountain Tire Tube	1992	0.133879965	0.125304948

Note This example is provided only to illustrate the usage of **BottomPercent**. Depending on the size of your data set, this query might take a long time to run.

Warning
The MDX functions for TOPPERCENT and BOTTOMPERCENT can generate unexpected results when the values used to calculate the percentage include negative numbers. This behavior does not affect the DMX functions. For more information, see BOTTOMPERCENT.

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

BottomSum

Returns, in order of increasing rank, the bottom-most rows of a table whose cumulative total is at least a specified value.

Syntax

BottomSum(<**`table expression`**>, <**`rank expression`**>, <**`sum`**>)

Applies To

An expression that returns a table, such as a <table column reference>, or a function that returns a table.

Return Type

<table expression>

Remarks

The **BottomSum** function returns the bottom-most rows in increasing order of rank. The rank is based on the evaluated value of the <rank expression> argument for each row, such that the sum of the <rank expression> values is at least the given total that is specified by the <sum> argument. **BottomSum** returns the smallest number of elements possible while still meeting the specified sum value.

Examples

The following example creates a prediction query against the Association model that you build by using the Basic Data Mining Tutorial.

To understand how **BottomSum** works, it might be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)
FROM
     [Association]
NATURAL PREDICTION JOIN
SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note

In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851
Water Bottle	2866	0.192620472	0.175205052
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **BottomSum** function takes the results of this query and returns the rows with the lowest values that sum to the specified count.

```
SELECT
BottomSum
    (
    Predict([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $PROBABILITY,
    .1)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **BottomSum** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **BottomSum** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $PROBABILITY to return rows that sum to at least 50% probability.

The third argument to the **BottomSum** function specifies the target sum, as a double. To get the rows for the lowest-count products that sum to 10 percent probability, you type .1.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Road Bottle Cage	1195	0.08…	0.07…
Mountain Bottle Cage	1367	0.09…	0.08…

Note This example is provided only to illustrate the usage of **BottomSum**. Depending on the size of your data set, this query might take a long time to run.

See Also

Functions (DMX)

Mapping Functions to Query Types (DMX)

BottomPercent (DMX)

Cluster

Returns the cluster that is most likely to contain the input case.

Syntax

```
Cluster()
```

Applies To

This function can be used only if the underlying data mining model supports clustering.

Return Type

The **Cluster** function does not require parameters.

The **Cluster** function returns a scalar value of a cluster name. However, if you use this function as an argument of another function, you must regard it as a <cluster column reference>.

Remarks

Cluster can also be used as a <cluster column reference> for a **PredictHistogram** function.

Examples

The following example uses a singleton query with the PredictHistogram (DMX) and **Cluster** functions to return the distance of the individual case from each cluster of the TM Clustering mining model and the probability that the individual case will exist in each cluster.

```
SELECT
  PredictHistogram(Cluster())
FROM
  [TM Clustering]
  NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)
Data Mining Extensions (DMX) Function Reference
Functions (DMX)
Mapping Functions to Query Types (DMX)

ClusterDistance

The **ClusterDistance** function returns the distance of the input case from the specified cluster, or if no cluster is specified, the distance of the input case from the most likely cluster.

Syntax

ClusterDistance([<**ClusterID expression**>])

Applies To

This function can be used only if the underlying data mining model supports clustering. The function can be used with any kind of clustering model (EM, K-Means, etc.), but the results differ depending on the algorithm.

Return Type

A scalar value.

Remarks

The **ClusterDistance** function returns the distance between the input case and the cluster that has the highest probability for that input case.

In case of K-Means clustering, since any case can belong to only one cluster, with a membership weight of 1.0, the cluster distance is always 0. However, in K-Means, each cluster is assumed to have a centroid. You can obtain the value of the centroid by querying or browsing the NODE_DISTRIBUTION nested table in the mining model content. For more information, see Mining Model Content for Clustering Models (Analysis Services - Data Mining).

In the case of the default EM clustering method, all the points inside the cluster are considered equally likely; therefore, by design there is no centroid for the cluster. The value of **ClusterDistance** between a particular case and a particular cluster N is calculated as follows:

Or:

Related Prediction Functions

Analysis Services provides the following additional functions for querying clustering models:

- Use the Cluster (DMX) function to return the most likely cluster.
- Use the ClusterProbability (DMX) function to get the probability that a case belongs to a particular cluster. This value serves as the inverse of the cluster distance.
- Use the PredictHistogram (DMX) function to return a histogram of the likelihood of the input case existing in each of the model's clusters.
- Use the PredictCaseLikelihood (DMX) function to return a measure from 0 to 1 that indicates how likely an input case is to exist considering the model learned by the algorithm.

Example1: Obtaining Cluster Distance to the Most Likely Cluster

The following example returns the distance from the specified case to the cluster that the case most likely belongs to.

```
SELECT
    ClusterDistance()
FROM
    [TM Clustering]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
    '2-5 Miles' AS [Commute Distance],
    'Graduate Degree' AS [Education],
    0 AS [Number Cars Owned],
    0 AS [Number Children At Home]) AS t
```

Example results:

Expression
0.0477390930705145

To find out which cluster this is, you can substitute `Cluster` for `ClusterDistance` in the preceding sample.

Example results:

$CLUSTER
Cluster 6

Example2: Obtaining Distance to a Specified Cluster

The following syntax uses the mining model content schema rowset to return the list of node IDs and node captions for the clusters in the mining model. You can then use the node caption as the cluster identifier argument in the **ClusterDistance** function.

```
SELECT NODE_UNIQUE_NAME, NODE_CAPTION
FROM <model>.CONTENT
WHERE NODE_TYPE = 5
```

Example results:

NODE_UNIQUE_NAME	NODE_CAPTION
001	Cluster 1
002	Cluster 2

The following syntax example returns the distance of the specified case from the cluster labeled Cluster 2.

```
SELECT
    ClusterDistance('Cluster 2')
AS [Cluster 2 Distance]
```

```
FROM [TM Clustering]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
    '2-5 Miles' AS [Commute Distance],
    'Graduate Degree' AS [Education],
    0 AS [Number Cars Owned],
    0 AS [Number Children At Home]) AS t
```

Example results:

Cluster 2 Distance
0.97008209236394

See Also

Cluster
Data Mining Extensions (DMX) Function Reference
Functions (DMX)
Mining Model Content for Clustering Models (Analysis Services - Data Mining)

ClusterProbability

Returns the probability that the input case belongs to the specified cluster.

Syntax

ClusterProbability([<**Node_Caption**>])

Applies To

This function can be used only if the underlying data mining model supports clustering.

Return Type

A scalar value.

Remarks

The following syntax uses the mining model content schema rowset to return the node captions that exist in the mining model.

```
SELECT NODE_CAPTION FROM <model>.CONTENT
```

For more information about using this syntax, see Mapping Functions to Query Types (DMX). For more information about the mining model content schema rowset, see DMSCHEMA_MINING_MODEL_CONTENT Rowset.

If a <node caption> is not specified, the function returns the probability that the input cases belong to the most likely cluster. Use the **Cluster** function to return the most likely cluster.

Examples

The following example returns the probability that the specified case exists in the cluster labeled Cluster 2.

```
SELECT
  ClusterProbability('Cluster 2')
From
  [TM Clustering]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Cluster

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

Exists

Returns **true** if the specified sub-query returns at least one row.

Syntax

```
EXISTS(<subquery>)
```

Arguments

subquery

A SELECT statement of the form SELECT * FROM <column name> [WHERE <predicate list>].

Result Type

Returns **true** if the result set returned by the subquery contains at least one row; otherwise, returns **false**.

Remarks

You can use the NOT keyword before EXISTS: for example, `WHERE NOT EXISTS (<subquery>)`.

The list of columns that you add to the sub-query argument of EXISTS is irrelevant; the function only checks for the existence of a row that meets the condition.

Examples

You can use EXISTS and NOT EXISTS to check for conditions in a nested table. This is useful when creating a filter that controls the data used to train or test a data mining model. For more information, see Creating Filters for Mining Models (Analysis Services - Data Mining).

The following example is based on the `[Association]` mining structure and mining model that you created in the Basic Data Mining Tutorial. The query returns only those cases where the customer purchased at least one patch kit.

```
SELECT * FROM [Association].CASES

WHERE EXISTS

(

SELECT * FROM [v Assoc Seq Line Numbers]

WHERE [[Model] = 'Patch kit'

)
```

Another way to view the same data that is returned by this query is to open the model in the Association viewer, right-click the itemset **Patch kit = Existing**, select the **Drill Through** option, and then select **Model Cases Only**.

See Also

Functions (DMX)

Model Filter Syntax and Examples (Analysis Services - Data Mining)

IsDescendant

Indicates whether the current node descends from the specified node.

Syntax

IsDescendant(<NodeID>)

Return Type

A Boolean type.

Remarks

IsDescendant is only used in Mapping Functions to Query Types (DMX) and SELECT FROM <model>.DIMENSION_CONTENT queries.

Examples

The following example returns all the cases that are descendents of the node that is specified in the IsDescendant function.

```
SELECT * FROM [TM Decision Tree].CONTENT

WHERE IsDescendant('00000000100')
```

See Also

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

IsInNode

Indicates whether the specified node contains the current case.

Syntax

IsInNode(<NodeID>)

Return Type

A Boolean type.

Remarks

IsInNode is only used in Mapping Functions to Query Types (DMX) and SELECT FROM <model>.SAMPLE_CASES (DMX) queries.

Examples

The following example returns all the cases that were used to create the model that is associated with the node that is specified in the IsInNode function.

```
Select * from [TM Decision Tree].Cases

WHERE IsInNode('0')
```

See Also

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

IsTestCase

Indicates whether a case is used as a test case for the specified data mining model or mining structure.

Syntax

IsTestCase()

Result Type

Returns **true** if the case is a part of the test data set; otherwise **false**.

Remarks

If you use the Data Mining Wizard to create a mining structure and related mining model, by default, 30 percent of the cases are set aside for use as a test data set. The remaining cases are used for training the data mining model. The same test data set can be used with all models that are based on that structure. However, if you use DMX to create the mining model, by default, all data is used to train the model, and no test set is created. To enable creation of a test data set, you must set the parameters of the WITH HOLDOUT clause.

You can determine whether a test set has been created on a particular mining structure by viewing the value of the **P:Microsoft.AnalysisServices.MiningStructure.HoldoutMaxCases** and **P:Microsoft.AnalysisServices.MiningStructure.HoldoutMaxPercent** properties.

Note
Drillthrough must be enabled on the model if you want to use the **IsTrainingCase** or **IsTestCase** functions to return details about the cases in a particular model. For more information, see How to: Enable Drillthrough for a Mining Model.

To return cases that are part of the training data set, use the function IsTrainingCase.

Examples

The following example uses the `Targeted Mailing` mining structure that is created in the Basic Data Mining Tutorial. The query returns all the cases in the structure that are used for testing.

```
SELECT *

FROM [Targeted Mailing].CASES

WHERE IsTestCase()
```

For more information about how to query cases used in data mining, see SELECT FROM <model>.CASES (DMX) and SELECT FROM <structure>.CASES.

See Also

Functions (DMX)

Querying Data Mining Models (Analysis Services - Data Mining)

Partitioning Data into Training and Testing Sets (Analysis Services - Data Mining)

IsTrainingCase

Indicates whether a case is used as a training case for the specified data mining model or mining structure.

Syntax

IsTrainingCase()

Result Type

Returns **true** if the case is a part of the training data set; otherwise **false**.

Remarks

If you use the Data Mining Wizard to create a mining structure and related mining model, by default, 30 percent of the cases are set aside for use as a test data set. The remaining cases in the data source that you specify are used to train the model. However, if you use Data Mining Extensions (DMX) to create the mining model, by default, all data is used to train the model, and no test set is created. To enable the creation of a test data set, you must set the parameters of the WITH HOLDOUT clause.

You can determine whether the data in a particular data mining structure has been partitioned into testing and training sets by viewing the value of the **P:Microsoft.AnalysisServices.MiningStructure.HoldoutMaxCases** and **P:Microsoft.AnalysisServices.MiningStructure.HoldoutMaxPercent** properties.

Note
Drillthrough must be enabled on the model if you want to use the **IsTrainingCase** or **IsTestCase** functions to return details about the cases in the model. For more information, see How to: Enable Drillthrough for a Mining Model.

To return cases that are part of the test data set, use the function IsTestCase.

Examples

The following example uses the clustering data mining model from the targeted mailing scenario in the Basic Data Mining Tutorial. The query returns only those cases that were used for training the mining model. Moreover, the training cases are restricted to customers younger than 40.

```
SELECT *

FROM [TM Clustering].CASES
```

```
WHERE IsTrainingCase()
AND [Age] <40
```

For other examples of how to query cases used in data mining, see SELECT FROM <model>.CASES (DMX) and SELECT FROM <structure>.CASES.

See Also

Partitioning Data into Training and Testing Sets (Analysis Services - Data Mining)

Functions (DMX)

Querying Data Mining Models (Analysis Services - Data Mining)

Lag

Returns the time slice between the date of the current case and the last date of the training set.

Syntax

Lag()

Return Type

A scalar value of the type integer.

Remarks

If the **Lag** function is used on a model where the KEY TIME column is located within a nested table, the function must be located within the sub-select of the statement.

Examples

The following example returns cases that fall within the last 12 months of the data that was used to train the model.

```
SELECT * FROM [Forecasting].CASES
WHERE Lag() < 12
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

Predict

The **Predict** function returns a predicted value, or set of values, for a specified column.

Syntax

Predict(<`scalar column reference`>, [`option1`], [`option2`], [`option n`], [INCLUDE_NODE_ID], `n`)

Predict(<`table column reference`>, [`option1`], [`option2`], [`option n`], [INCLUDE_NODE_ID], `n`)

Applies To

Either a scalar column reference or a table column reference.

Return Type

<scalar column reference>

or

<table column reference>

The return type depends on the type of column to which this function is applied.

Note
INCLUSIVE, EXCLUSIVE, INPUT_ONLY, and INCLUDE_STATISTICS apply only for a table column reference, and EXCLUDE_NULL and INCLUDE_NULL apply only for a scalar column reference.

Remarks

Options include EXCLUDE_NULL (default), INCLUDE_NULL, INCLUSIVE, EXCLUSIVE (default), INPUT_ONLY, and INCLUDE_STATISTICS.

Note
For time series models, the **Predict** function does not support INCLUDE_STATISTICS.

The INCLUDE_NODE_ID parameter returns the $NODEID column in the result. NODE_ID is the content node on which the prediction is executed for a particular case. This parameter is optional when using **Predict** on table columns.

The n parameter applies to table columns. It sets the number of rows that are returned based on the type of prediction. If the underlying column is sequence, it calls the **PredictSequence** function. If the underlying column is time series, it calls the **PredictTimeSeries** function. For associative types of prediction, it calls the **PredictAssociation** function.

The **Predict** function supports polymorphism.

The following alternative abbreviated forms are frequently used:

- [Gender] is an alternative for **Predict**([Gender], EXCLUDE_NULL).
- [Products Purchases] is an alternative for **Predict**([Products Purchases], EXCLUDE_NULL, EXCLUSIVE).

  Note

The return type of this function is itself regarded as a column reference. This means that the **Predict** function can be used as an argument in other functions that take a column reference as an argument (except for the **Predict** function itself).

Passing INCLUDE_STATISTICS to a prediction on a table-valued column adds the columns **$Probability** and **$Support** to the resulting table. These columns describe the probability of existence for the associated nested table record.

Examples

The following example uses the **Predict** function to return the four products in the Adventure Works database that are most likely to be sold together. Because the function is predicting against an association rules mining model, it automatically uses the **PredictAssociation** function as described earlier.

```
SELECT
    Predict([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,4)
FROM     [Association]
```

Sample results:

This query returns a single row of data with one column, `Expression`, but that column contains the following nested table.

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016331743	0.252695851192499
Water Bottle	2866	0.192620471805901	0.175205052318795
Patch Kit	2113	0.142012232004839	0.132389356196586
Mountain Tire Tube	1992	0.133879965051415	0.125304947722259

See Also

[Mapping Functions to Query Types (DMX)]

[Functions (DMX)]

[Mapping Functions to Query Types (DMX)]

PredictAdjustedProbability

Returns the adjusted probability of a specified state.

Syntax

PredictAdjustedProbability(<**`scalar column reference`**>, [<**`predicted state`**>])

Applies To

A scalar column.

Return Type

A scalar value.

Remarks

If the predicted state is omitted, the state that has the highest predictable probability is used, excluding the missing states bucket. To include the missing states bucket, set the <predicted state> to **INCLUDE_NULL**.

To return the adjusted probability for the missing states, set the <predicted state> to NULL.

The **PredictAdjustedProbability** function is a Microsoft SQL Server Analysis Services extension to the Microsoft OLE DB for Data Mining specification.

Examples

The following example uses a natural prediction join to determine if an individual is likely to be a bike buyer based on the TM Decision Tree mining model, and also determines the adjusted probability for the prediction.

```
SELECT
  [Bike Buyer],
  PredictAdjustedProbability([Bike Buyer]) AS [Adjusted Probability]
From
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictAssociation

Predicts associative membership.

Note
This is different from creating a prediction on an association model.

Syntax

PredictAssociation(<**`table column reference`**>, **`option1`**, **`option2`**, **`n`** ...)

Applies To

Classification algorithms and clustering algorithms that contain predictable nested tables. Classification algorithms include the Microsoft Decision Trees, Microsoft Naive Bayes, and Microsoft Neural Network algorithms.

Return Type

<table expression>

Remarks

The options for the **PredictAssociation** function include EXCLUDE_NULL, INCLUDE_NULL, INCLUSIVE, EXCLUSIVE (default), INPUT_ONLY, INCLUDE_STATISTICS, and INCLUDE_NODE_ID.

Note
INCLUSIVE, EXCLUSIVE, INPUT_ONLY, and INCLUDE_STATISTICS apply only for a table column reference, and EXCLUDE_NULL and INCLUDE_NULL apply only for a scalar column reference.

INCLUDE_STATISTICS only returns **$Probability** and **$AdjustedProbability**.

If the numeric parameter n is specified, the **PredictAssociation** function returns the top most likely values based on the probability:

```
PredictAssociation(colref, [$AdjustedProbability], n)
```

If you include **$AdjustedProbability**, the statement returns the top n values based on the **$AdjustedProbability**.

Examples

The following example uses the **PredictAssociation** function to return the four products in the Adventure Works database that are most likely to be sold together.

```
SELECT

  PredictAssociation([Association].[v Assoc Seq Line Items],4)

From

  [Association]
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictCaseLikelihood

This function returns the likelihood that an input case will fit in the existing model. Used only with clustering models.

Syntax

```
PredictCaseLikelihood([NORMALIZED|NONNORMALIZED])
```

Arguments

NORMALIZED

Return value contains the probability of the case within the model divided by the probability of the case without the model.

NONNORMALIZED

Return value contains the raw probability of the case, which is the product of the probabilities of the case attributes.

Applies To

Models that are built by using the Microsoft Clustering and Microsoft Sequence Clustering algorithms.

Return Type

Double-precision floating point number between 0 and 1. A number closer to 1 indicates that the case has a higher probability of occurring in this model. A number closer to 0 indicates that the case is less likely to occur in this model.

Remarks

By default, the result of the **PredictCaseLikelihood** function is normalized. Normalized values are typically more useful as the number of attributes in a case increase and the differences between the raw probabilities of any two cases become much smaller.

The following equation is used to calculate the normalized values, given x and y:

- x = likelihood of the case based on the clustering model
- y = Marginal case likelihood, calculated as the log likelihood of the case based on counting the training cases
-

Normalized

Examples

The following example returns the likelihood that the specified case will occur within the clustering model, which is based on the Adventure Works DW database.

```
SELECT
  PredictCaseLikelihood() AS Default_Likelihood,
  PredictCaseLikelihood(NORMALIZED) AS Normalized_Likelihood,
  PredictCaseLikelihood(NONNORMALIZED) AS Raw_Likelihood,
FROM
  [TM Clustering]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

Expected results:

Default_Likelihood	Normalized_Likelihood	Raw_Likelihood
6.30672792729321E-08	6.30672792729321E-08	9.5824454056846E-48

The difference between these results demonstrates the effect of normalization. The raw value for **CaseLikelihood** suggests that the probability of the case is about 20 percent; however, when you normalize the results, it becomes apparent that the likelihood of the case is very low.

See Also

Mapping Functions to Query Types (DMX)

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictHistogram

Returns a table that represents a histogram for the prediction of a given column.

Syntax

PredictHistogram(**`<scalar column reference>`** | **`<cluster column reference>`**)

Applies To

A scalar column reference or a cluster column reference. Can be used with all algorithm types except the Microsoft Association algorithm.

Return Type

A table.

Remarks

A histogram generates statistics columns. The column structure of the returned histogram depends on the type of column reference that is used with the **PredictHistogram** function.

Scalar Columns

For a <scalar column reference>, the histogram that the **PredictHistogram** function returns consists of the following columns:

- The value that is being predicted.
- **$Support**
- **$Probability**
- **$ProbabilityVariance**

  Microsoft data mining algorithms do not support **$ProbabilityVariance**. This column always contains 0 for Microsoft algorithms.
- **$ProbabilityStdev**

  Microsoft data mining algorithms do not support **$ProbabilityStdev**. This column always contains 0 for Microsoft algorithms.
- **$AdjustedProbability**

  The **$AdjustedProbability** column is an Analysis Services extension to the Microsoft OLE DB for Data Mining specification.

Cluster Columns

The histogram that the **PredictHistogram** function returns for a <cluster column reference> consists of the following columns:

- **$Cluster** (represents the cluster name)
- **$Distance**
- **$Probability**

Examples

The following example returns the predicted state of the Bike Buyer column in a singleton query. The query also returns the top two most likely states of the Bike Buyer attribute, based on the adjusted probability obtained by using the **PredictHistogram** function.

```
SELECT
  [TM Decision Tree].[Bike Buyer],
  TopCount(PredictHistogram([Bike Buyer]),$AdjustedProbability,3)
From
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)

ClusterProbability

PredictAdjustedProbability

PredictProbability

PredictStdev

PredictSupport

PredictVariance

Data Mining Algorithms

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictNodeId

Returns the Node_ID of the node to which the case is classified.

Syntax

PredictNodeId(<**`scalar column reference`**>)

Applies To

A scalar column.

Return Type

<scalar expression>

Examples

The following example returns whether the specified individual is likely to buy a bicycle, and also returns the nodeID of the node that they are most likely to be part of.

```
SELECT
  [Bike Buyer],
  PredictNodeId([Bike Buyer])
From
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

You could then use the following statement to determine what is contained within the node:

```
SELECT
  NODE_CAPTION
FROM
  [TM Decision Tree].CONTENT
WHERE NODE_UNIQUE_NAME= '00000000100'
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictProbability

Returns the probability for a specified state.

Syntax

PredictProbability(<**`scalar column reference`**>, [<**`predicted state`**>])

Applies To

A scalar column.

Return Type

A scalar value.

Remarks

If the predicted state is omitted, the state that has the highest probability is used, excluding the missing states bucket. To include the missing states bucket, set the <predicted state> to **INCLUDE_NULL**. To return the probability for the missing states, set the <predicted state> to NULL.

Note
Some mining models do not provide probability values and therefore cannot use this function. In addition, the probability values for any particular target value are calculated differently or might have a different interpretation depending on the model type that you are querying. For more information about how probability is calculated for a particular model type, see the individual algorithm topic in Mining Model Content (Analysis Services - Data Mining).

Examples

The following example uses a natural prediction join to determine whether an individual is likely to be a bike buyer based on the TM Decision Tree mining model, and also determines the probability for the prediction. In this example, there are two PredictProbability functions, one for each possible value. If you omit this argument, the function returns the probability for the most likely value.

```
SELECT
  [Bike Buyer],
  PredictProbability([Bike Buyer], 1) AS [Bike Buyer = Yes],
  PredictProbability([Bike Buyer], 0) AS [Bike Buyer = No]
FROM [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

Example results:

Bike Buyer	Bike Buyer = Yes	Bike Buyer = No
1	0.867074195848097	0.132755556974282

See Also

Data Mining Functions Reference)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictSequence

Predicts future sequence values for a specified set of sequence data.

Syntax

PredictSequence(<**table column reference**>)

PredictSequence(<**table column reference**, n>)

PredictSequence(<**table column reference**, **n-start**, **n-end**>)

Return Type

A <table expression>.

Remarks

If the n parameter is specified, it returns the following values:

- If n is greater than zero, the most likely sequence values in the next n steps.
- If both n-start and n-end are specified, the sequence values from n-start to n-end.

Examples

The following example returns a sequence of the five products that are most likely to be purchased by a customer in the Adventure Works DW Multidimensional 2012 database based on the Sequence Clustering mining model.

```
SELECT
  PredictSequence([Sequence Clustering].[v Assoc Seq Line Items],5)
From
  [Sequence Clustering]
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictStdev

Returns the predicted standard deviation for the specified column.

Syntax

PredictStdev(<**`scalar column reference`**>)

Applies To

A scalar column.

Return Type

A scalar value of the type that is specified by <scalar column reference>.

Remarks

If the column reference is discrete, **PredictStdev** returns 0 because the standard deviation cannot be calculated from discrete values.

Examples

The following example uses a natural prediction join to determine whether an individual is likely to be a bike buyer based on the TM Decision Tree mining model, and also determines the standard deviation for the prediction.

```
SELECT
  [Bike Buyer],
  PredictStdev([Bike Buyer]) AS [Standard Deviation]
FROM
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictSupport

Returns the support value for a specified state.

Syntax

PredictSupport(<**scalar column reference**>, [<**predicted state**>])

Applies To

A scalar column.

Return Type

A scalar value of the type that is specified by <scalar column reference>.

Remarks

If the predicted state is omitted, the state that has the highest predictable probability is used, excluding the missing states bucket. To include the missing states bucket, set the <predicted state> to **INCLUDE_NULL**.

To return the support for the missing states, set the <predicted state> to NULL.

Note

The support values are calculated differently or might have a different interpretation depending on the model type that you are querying. For more information about how support is calculated for any particular model type, see the individual algorithm type in Mining Model Content (Analysis Services - Data Mining).

Examples

The following example uses a singleton query to predict whether an individual will be a bike buyer, and also determines the support for the prediction based on the TM Decision Tree mining model.

```
SELECT
  [Bike Buyer],
  PredictSupport([Bike Buyer]) AS [Support],
From
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

PredictTimeSeries

Returns predicted future values for time series data. Time series data is continuous and can be stored in a nested table or in a case table. The **PredictTimeSeries** function always returns a nested table.

Syntax

`PredictTimeSeries`(**`<table column reference>`**)

`PredictTimeSeries`(**`<table column reference>`**, **`n`**)

`PredictTimeSeries`(**`<table column reference>`**, **`n-start`**, **`n-end`**)

`PredictTimeSeries`(**`<scalar column reference>`**)

`PredictTimeSeries`(**`<scalar column reference>`**, **`n`**)

PredictTimeSeries(**`<scalar column reference>`**, **`n-start`**, **`n-end`**)

`PredictTimeSeries`(**`<table column reference>`**, **`n, REPLACE_MODEL_CASES`** | EXTEND_MODEL_CASES) **`PREDICTION JOIN <source query>`**

PredictTimeSeries(**`<table column reference>`**, **`n-start`**, **`n-end,`** **`REPLACE_MODEL_CASES`** | EXTEND_MODEL_CASES) **`PREDICTION JOIN <source query>`**

PredictTimeSeries(**`<scalar column reference>`**, **`n, REPLACE_MODEL_CASES`** | EXTEND_MODEL_CASES) **`PREDICTION JOIN <source query>`**

PredictTimeSeries(**`<scalar column reference>`**, **`n-start`**, **`n-end,`** **`REPLACE_MODEL_CASES`** | EXTEND_MODEL_CASES) **`PREDICTION JOIN <source query>`**

Arguments

<table column reference>, <scalar column referenc>

Specifies the name of the column to predict. The column can contain either scalar or tabular data.

n

Specifies the number of next steps to predict. If a value is not specified for n, the default is 1.

n cannot be 0. The function returns an error if you do not make at least one prediction.

n-start, n-end

Specifies a range of time series steps.

n-start must be an integer and cannot be 0.

n-end must be an integer greater than n-start.

<source query>

Defines the external data that is used for making predictions.

REPLACE_MODEL_CASES | EXTEND_MODEL_CASES

Indicates how to handle new data.

REPLACE_MODEL_CASES specifies that the data points in the model should be replaced with new data. However, predictions are based on the patterns in the existing mining model.

EXTEND_MODEL_CASES specifies that the new data should be added to the original training data set. Future predictions are made on the composite data set only after the new data has been used up.

These arguments can be used only when new data is added by using a PREDICTION JOIN statement. If you use a PREDICTION JOIN query and do not specify an argument, the default is EXTEND_MODEL_CASES.

Return Type

A <table expression>.

Remarks

The Microsoft Time Series algorithm does not support historical prediction when you use the PREDICTION JOIN statement to add new data.

In a PREDICTION JOIN, the prediction process always starts at the time step immediately after the end of the original training series. This is true even if you add new data. Therefore, the n parameter and n-start parameter values must be an integer greater than 0.

Note

The length of the new data does not affect the starting point for prediction. Therefore, if you want to add new data and also make new predictions, make sure that you either set the prediction start point to a value greater than the length of the new data, or extend the prediction end point by the length of the new data.

Examples

The following examples show how to make predictions against an existing time series model:

- The first example shows how to make a specified number of predictions based on the current model.

- The second example shows how to use the REPLACE_MODEL_CASES parameter to apply the patterns in the specified model to a new set of data.
- The third example shows how to use the EXTEND_MODEL_CASES parameter to update a mining model with fresh data.

To learn more about working with time series models, see the data mining tutorial, Lesson 2: Building a Forecasting Scenario (Intermediate Data Mining Tutorial) and Time Series Prediction DMX Tutorial.

Note
You might obtain different results from your model; the results of the examples below are provided only to illustrate the result format.

Example 1: Predicting a Number of Time Slices

The following example uses the **PredictTimeSeries** function to return a prediction for the next three time steps, and restricts results to the M200 series in the Europe and Pacific regions. In this particular model, the predictable attribute is Quantity, so you must use `[Quantity]` as the first argument to the **PredictTimeSeries** function.

```
SELECT FLATTENED
    [Forecasting].[Model Region],
    PredictTimeSeries([Forecasting].[Quantity],3)AS t
FROM
    [Forecasting]
WHERE [Model Region] = 'M200 Europe'
OR [Model Region] = 'M200 Pacific'
```

Expected results:

Model Region	t.$TIME	t.Quantity
M200 Europe	7/25/2008 12:00:00 AM	121
M200 Europe	8/25/2008 12:00:00 AM	142
M200 Europe	9/25/2008 12:00:00 AM	152
M200 Pacific	7/25/2008 12:00:00 AM	46
M200 Pacific	8/25/2008 12:00:00 AM	44
M200 Pacific	9/25/2008 12:00:00 AM	42

In this example, the FLATTENED keyword has been used to make the results easier to read. If you do not use the FLATTENED keyword and instead return a hierarchical rowset, this query returns two columns. The first column contains the value for [ModelRegion], and the second column contains a nested table with two columns: $TIME, which shows the time slices that are being predicted, and Quantity, which contains the predicted values.

Example 2: Adding New Data and Using REPLACE_MODEL_CASES

Suppose you find that the data was incorrect for a particular region, and want to use the patterns in the model, but to adjust the predictions to match the new data. Or, you might find that another region has more reliable trends and you want to apply the most reliable model to data from a different region.

In such scenarios, you can use the REPLACE_MODEL_CASES parameter and specify a new set of data to use as historical data. That way, the projections will be based on the patterns in the specified model, but will continue smoothly from the end of the new data points. For a complete walkthrough of this scenario, see Adding an Aggregated Forecasting Model (Intermediate Data Mining Tutorial).

The following PREDICTION JOIN query illustrates the syntax for replacing data and making new predictions. For the replacement data, the example retrieves the value of the Amount and Quantity columns and multiplies each by two:

```
SELECT [Forecasting].[Model Region],
    PredictTimeSeries([Forecasting].[Quantity], 3, REPLACE_MODEL_CASES)
FROM
    [Forecasting]
PREDICTION JOIN
  OPENQUERY([Adventure Works DW Multidimensional 2012],
    'SELECT [ModelRegion],
    ([Quantity] * 2) as Quantity,
    ([Amount] * 2) as Amount,
      [ReportingDate]
    FROM [dbo].vTimeSeries
    WHERE ModelRegion = N''M200 Pacific''
    ') AS t
ON
  [Forecasting].[Model Region] = t.[ Model Region] AND
[Forecasting].[Reporting Date] = t.[ReportingDate] AND
[Forecasting].[Quantity] = t.[Quantity] AND
[Forecasting].[Amount] = t.[Amount]
```

The following table compares the results of prediction.

Original predictions			**Updated predictions**		
M200 Pacific	7/25/2008 12:00:00 AM	46	M200 Pacific	7/25/2008 12:00:00 AM	91
M200 Pacific	8/25/2008 12:00:00 AM	44	M200 Pacific	8/25/2008 12:00:00 AM	89
M200 Pacific	9/25/2008 12:00:00 AM	42	M200 Pacific	9/25/2008 12:00:00 AM	84

Example 3: Adding New Data and Using EXTEND_MODEL_CASES

Example 3 illustrates the use of the EXTEND_MODEL_CASES option to provide new data, which is added to the end of an existing data series. Rather than replacing the existing data points, the new data is added onto the model.

In the following example, the new data is provided in the SELECT statement that follows NATURAL PREDICTION JOIN. You can supply multiple rows of new input with this syntax, but each new row of input must have a unique time stamp:

```
SELECT [Model Region],
    PredictTimeSeries([Forecasting].[Quantity], 5, EXTEND_MODEL_CASES)
FROM
    [Forecasting]
NATURAL PREDICTION JOIN
    (SELECT
        1 as [Reporting Date],
        10 as [Quantity],
        'M200 Europe' AS [Model Region]
    UNION SELECT
        2 as [Reporting Date],
        15 as [Quantity],
        'M200 Europe' AS [Model Region]
) AS T
```

```
WHERE ([Model Region] = 'M200 Europe'
 OR [Model Region] = 'M200 Pacific')
```

Because the query uses the EXTEND_MODEL_CASES option, Analysis Services takes the following actions for its predictions:

- Increases the total size of the training cases by adding the two new months of data to the model.
- Starts the predictions at the end of the previous case data. Therefore, the first two predictions represent the new actual sales data that you just added to the model.
- Returns new predictions for the remaining three time slices based on the newly expanded model.

The following table lists the results of the Example 2 query. Notice that the first two values returned for M200 Europe are exactly the same as the new values that you provided. This behavior is by design; if you want to start predictions after the end of the new data, you must specify a starting and ending time step. For an example of how to do this, see Lesson 5: Extending the Time Series Model.

Also, notice that you did not supply new data for the Pacific region. Therefore, Analysis Services returns new predictions for all five time slices.

Quantity	**EXTEND_MODEL_CASES**	
	$TIME	**Quantity**
M200 Europe	7/25/2008 0:00	10
	8/25/2008 0:00	15
	9/25/2008 0:00	72
	10/25/2008 0:00	69
	11/25/2008 0:00	68
	$TIME	**Quantity**
M200 Pacific	7/25/2008 0:00	46
	8/25/2008 0:00	44

Quantity	EXTEND_MODEL_CASES
	9/25/2008 0:00 | 42 10/25/2008 0:00 | 42 11/25/2008 0:00 | 38

Example 4: Returning Statistics in a Time Series Prediction

The **PredictTimeSeries** function does not support INCLUDE_STATISTICS as a parameter. However, the following query can be used to return the prediction statistics for a time series query. This approach can also be used with models that have nested table columns.

In this particular model, the predictable attribute is Quantity, so you must use `[Quantity]` as the first argument to the **PredictTimeSeries** function. If your model uses a different predictable attribute, you can substitute a different column name.

```
SELECT FLATTENED [Model Region],
(SELECT
     $Time,
     [Quantity] as [PREDICTION],
     PredictVariance([Quantity]) AS [VARIANCE],
     PredictStdev([Quantity]) AS [STDEV]
FROM
      PredictTimeSeries([Quantity], 3) AS t
) AS t
FROM Forecasting
WHERE [Model Region] = 'M200 Europe'
OR [Model Region] = 'M200 North America'
```

Sample results:

Model Region	t.$TIME	t.PREDICTION	t.VARIANCE	t.STDEV
M200	7/25/2008	121	11.6050581415597	3.40661975300439

Model Region	t.$TIME	t.PREDICTION	t.VARIANCE	t.STDEV
Europe	12:00:00 AM			
M200 Europe	8/25/2008 12:00:00 AM	142	10.678201866621	3.26775180615374
M200 Europe	9/25/2008 12:00:00 AM	152	9.86897842568614	3.14149302493037
M200 North America	7/25/2008 12:00:00 AM	163	1.20434529288162	1.20434529288162
M200 North America	8/25/2008 12:00:00 AM	178	1.65031343900634	1.65031343900634
M200 North America	9/25/2008 12:00:00 AM	156	1.68969399185442	1.68969399185442

Note
The FLATTENED keyword was used in this example to make the results easier to present in a table; however, if your provider supports hierarchical rowsets you can omit the FLATTENED keyword. If you omit the FLATTENED keyword, the query returns two columns, the first column containing the value that identifies the `[Model Region]` data series, and the second column containing the nested table of statistics.

See Also

Data Mining Extensions (DMX) Function Reference

Querying a Time Series Model (Analysis Services - Data Mining)

Predict (DMX)

PredictVariance

Returns the variance of a specified column.

Syntax

PredictVariance(<**`scalar column reference`**>)

Applies To

A scalar column.

Return Type

A scalar value of the type that is specified by <scalar column reference>.

Remarks

If the column reference is discrete, **PredictVariance** returns 0 because the variance cannot be calculated from discrete values.

Examples

The following example uses a natural prediction join to determine if an individual is likely to be a bike buyer based on the TM Decision Tree mining model, and also determines the variance for the prediction.

```
SELECT
  [Bike Buyer],
  PredictVariance([Bike Buyer]) AS [Variance]
FROM
  [TM Decision Tree]
NATURAL PREDICTION JOIN
(SELECT 28 AS [Age],
  '2-5 Miles' AS [Commute Distance],
  'Graduate Degree' AS [Education],
  0 AS [Number Cars Owned],
  0 AS [Number Children At Home]) AS t
```

See Also

Mapping Functions to Query Types (DMX)
Functions (DMX)
Mapping Functions to Query Types (DMX)

RangeMax

Returns the upper end of the predicted bucket that is discovered for a discretized column.

Syntax

RangeMax(<**`scalar column reference`**>)

Applies To

Scalar columns.

Return Type

A scalar value.

Remarks

The **RangeMax** function can be used in RangeMin queries. When used with this type of query, the scalar column reference can contain continuous or discrete columns that are either predictable or input.

When used with PREDICTION JOIN, the **RangeMin**, **RangeMid**, and **RangeMax** functions return the actual boundary values of the specified bucket. For example, if you perform a prediction on a discretized column, the query returns the predicted bucket number in the discretized column. The **RangeMin**, **RangeMid**, and **RangeMax** functions describe the bucket that the prediction specifies. When the **RangeMax** function is used with a PREDICTION JOIN statement, the scalar column reference can only contain discrete, predictable columns.

Examples

The following example returns the minimum, maximum, and average values for the Yearly Income continuous column in the Decision Tree mining model.

```
SELECT DISTINCT

    RangeMin([Yearly Income]) AS [Bucket Minimum],

    RangeMid([Yearly Income]) AS [Bucket Average],

    RangeMax([Yearly Income]) AS [Bucket Maximum]

FROM [TM Decision Tree]
```

See Also

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

RangeMid (DMX)

RangeMin (DMX)

RangeMid

Returns the midpoint of the predicted bucket that is discovered for a discretized column.

Syntax

RangeMid(<**`scalar column reference`**>)

Applies To

Discretized scalar columns.

Return Type

A scalar value.

Remarks

When used with RangeMin, the **RangeMin**, **RangeMid**, and **RangeMax** functions return the actual boundary values of the specified bucket. For example, if you perform a prediction on a discretized column, the query returns the predicted bucket number in the discretized column. The **RangeMin**, **RangeMid**, and **RangeMax** functions describe the bucket that the prediction specifies. When the **RangeMid** function is used with a PREDICTION JOIN statement, the scalar column reference can only contain discrete, predictable columns.

Examples

The following example returns the minimum, maximum, and average values for the Yearly Income continuous column in a TM Decision Tree mining model.

```
SELECT DISTINCT

    RangeMin([Yearly Income]) AS [Bucket Minimum],

    RangeMid([Yearly Income]) AS [Bucket Average],

    RangeMax([Yearly Income]) AS [Bucket Maximum]

FROM [TM Decision Tree]
```

See Also

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

RangeMax (DMX)

RangeMin (DMX)

RangeMin

Returns the lower end of the predicted bucket that is discovered for a discretized column.

Syntax

RangeMin(<**scalar column reference**>)

Applies To

Scalar columns.

Return Type

A scalar value.

Remarks

The **RangeMin** function can be used in RangeMid queries. When used with this type of query, the scalar column reference can contain continuous or discrete columns that are either predictable or input.

When used with PREDICTION JOIN, the **RangeMin**, **RangeMid**, and **RangeMax** functions return the actual boundary values of the specified bucket. For example, if you perform a prediction on a discretized column, the query returns the predicted bucket number in the discretized column. The **RangeMin**, **RangeMid**, and **RangeMax** functions describe the bucket that the prediction specifies. When the **RangeMin** function is used with a PREDICTION JOIN statement, the scalar column reference can only contain discrete, predictable columns.

Examples

The following example returns the minimum, maximum, and average values for the Yearly Income continuous column in the Decision Tree mining model.

```
SELECT DISTINCT
    RangeMin([Yearly Income]) AS [Bucket Minimum],
    RangeMid([Yearly Income]) AS [Bucket Average],
    RangeMax([Yearly Income]) AS [Bucket Maximum]
FROM [TM Decision Tree]
```

See Also

Data Mining Extensions (DMX) Function Reference

Functions (DMX)

Mapping Functions to Query Types (DMX)

RangeMax (DMX)

RangeMid (DMX)

StructureColumn

Returns the value of the structure column corresponding to the specified case, or the table value for a nested table in the specified case.

Syntax

StructureColumn('structure column name')

Arguments

structure-column-name.

The name of a case or nested table mining structure column.

Result Type

The type that is returned depends on the type of the column that is referenced in the <structure column name> parameter. For example, if the mining structure column that is referenced contains a scalar value, the function returns a scalar value.

If the mining structure column that is referenced is a nested table, the function returns a table value. The returned table value can be used in the FROM clause of a sub-SELECT statement.

Remarks

This function is polymorphic and can be used anywhere in a statement that allows expressions, including a SELECT expression list, a WHERE condition expression, and an ORDER BY expression.

The name of the column in the mining structure is a string value and as such must be enclosed in single quotation marks: for example, `StructureColumn('`**column 1**`')`. If there are multiple columns that have the same name, the name is resolved in the context of the enclosing SELECT statement.

The results that are returned from a query using the **StructureColumn** function are affected by the presence of any filters on the model. That is to say, the model filter controls the cases that are included in the mining model. Therefore, a query on the structure column can return only those cases that were used in the mining model. See the Examples section of this topic for a code sample that shows the effect of mining model filters on both case tables and a nested table.

For more information about how to use this function in a DMX SELECT statement, see SELECT FROM <model>.CASES (DMX) or SELECT FROM <structure>.CASES.

Error Messages

The following security error is raised if the user does not have drillthrough permission on the parent mining structure:

The '%{user/}' user does not have permission to drill through to the parent mining structure of the ‘%{model/}’ mining model.

The following error message is raised if an invalid structure column name is specified:

The ‘%{structure-column-name/}’ mining structure column was not found in the ‘%{structure/}’ parent mining structure in the current context (line %{line/}, column %{column/}).

Examples

We will use the following mining structure for these examples. Note that the mining structure contains two nested table columns, `Products` and `Hobbies`. The nested table in the `Hobbies` column has a single column that is used as the key for the nested table. The nested table in the `Products` column is a complex nested table that has both a key column and other columns used for input. The following examples illustrate how a data mining structure can be designed to include many different columns, even though a model may not use every column. Some of these columns may not be useful at the model level for generalizing patterns, but may be very useful for drillthrough.

```
CREATE MINING STRUCTURE [MyStructure]
(
CustomerName TEXT KEY,
Occupation TEXT DISCRETE,
Age LONG CONTINUOUS,
MaritalStatus TEXT DISCRETE,
Income LONG CONTINUOUS,
Products  TABLE
 (
    ProductNameTEXT KEY,
    Quantity LONG CONTINUOUS,
    OnSale BOOLEAN  DISCRETE
)
 Hobbies  TABLE
 (
    Hobby KEY
 ))
```

Next, create a mining model based on the structure you just created, using the following example code:

```
ALTER MINING STRUCTURE [MyStructure] ADD MINING MODEL [MyModel] (
CustomerName,
Age,
MaritalStatus,
Income PREDICT,
Products
(
ProductName
) WITH FILTER(NOT OnSale)
) USING Microsoft_Decision_Trees
WITH FILTER(EXISTS (Products))
```

Sample Query 1: Returning a Column from the Mining Structure

The following sample query returns the columns `CustomerName` and `Age`, which are defined as being part of the mining model. However, the query also returns the column `Age`, which is part of the structure but not part of the mining model.

```
SELECT CustomerName, Age, StructureColumn('Occupation') FROM
MyModel.CASES
WHERE Age > 30
```

Note that the filtering of rows to restrict cases to customers over the age of 30 takes place at the level of the model. Therefore, this expression would not return cases that are included in the structure data but are not used by the model. Because the filter condition used to create the model (`EXISTS (Products)`) restricts cases to only those customers who purchased products, there might be cases in the structure that are not returned by this query.

Sample Query 2: Applying a Filter to the Structure Column

The following sample query not only returns the model columns `CustomerName` and `Age`, and the nested table `Products`, but also returns the value of the column `Quantity` in the nested table, which is not part of the model.

```
SELECT CustomerName, Age,
(SELECT ProductName, StructureColumn('Quantity') FROM Products) FROM
MA.CASES
WHERE StructureColumn('Occupation') = 'Architect'
```

Note that, in this example, a filter is applied to the structure column to restrict the cases to customers whose occupation is 'Architect' (`WHERE StructureColumn('Occupation') = 'Architect'`). Because the model filter condition is always applied to the cases when the model is created, only cases that contain at least one qualifying row in the `Products` table are included in the model cases. Therefore, both the filter on the nested table `Products` and the filter on the case (`('Occupation')`)are applied.

Sample Query 3: Selecting Columns from a Nested Table

The following sample query returns the names of customers who were used as training cases from the model. For each customer, the query also returns a nested table that contains the purchase details. Although the model includes the `ProductName` column, the model does not use the value of the `ProductName` column. The model only checks if the product was purchased at regular (`NOT OnSale`) price. This query not only returns the product name, but also returns the quantity purchased, which is not included in the model.

```
SELECT CustomerName,
(SELECT ProductName, StructureColumn('Quantity')FROM Products)
FROM MyModel.CASES
```

Note that you cannot return either the `ProductName` column or the `Quantity` column unless drillthrough is enabled on the mining model.

Sample Query 4: Filtering on and Returning Nested Table Columns

Thee following sample query returns the case and nested table columns that are included in the mining structure but not in the model. The model is already filtered on the presence of `OnSale` products, but this query adds a filter on the mining structure column, `Quantity`:

```
SELECT CustomerName, Age, StructureColumn('Occupation'),
(SELECT ProductName, StructureColumn('Quantity') FROM Products)
FROM MyModel.CASES
WHERE EXISTS (SELECT * FROM Products WHERE
StructureColumn('Quantity')>1)
```

See Also

Mapping Functions to Query Types (DMX)
Functions (DMX)
Mapping Functions to Query Types (DMX)

TopCount

Returns the specified number of top-most rows in decreasing order of rank as specified by an expression.

Syntax

TopCount(<**`table expression`**>, <**`rank expression`**>, <**`count`**>)

Applies To

An expression that returns a table, such as a <table column reference>, or a function that returns a table.

Return Type

<table expression>

Remarks

The value that is supplied by the <rank expression> argument determines the decreasing order of rank for the rows that are supplied in the <table expression> argument, and the number of top-most rows that is specified in the <count> argument is returned.

The **TopCount** function was originally introduced to enable associative predictions and in general, produces the same results as a statement that includes **SELECT TOP** and **ORDER BY** clauses. You will obtain better performance for associative predictions if you use the **Predict (DMX)** function, which supports specification of a number of predictions to return.

However, there are situations where you might still need to use **TopCount**. For example, DMX does not support the **TOP** qualifier in a sub-select statement. The PredictHistogram (DMX) function also does not support the addition of **TOP**.

Examples

The following examples are prediction queries against the Association model that you build by using the Basic Data Mining Tutorial. The queries return the same results, but the first example uses **TopCount**, and the second example uses the **Predict** function.

To understand how **TopCount** works, it may be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)

FROM

     [Association]

NATURAL PREDICTION JOIN

SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note

In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851
Water Bottle	2866	0.192620472	0.175205052

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **TopCount** function takes the results of this query and returns the specified number of the smallest-valued rows.

```
SELECT
TopCount
    (
    Predict ([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $SUPPORT,
    3)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **TopCount** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **TopCount** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $SUPPORT to rank the results.

The third argument to the **TopCount** function specifies the number of rows to return, as an integer. To get the top three products, as ordered by $SUPPORT, you type 3.

Example results:

Model	**$SUPPORT**	**$PROBABILITY**	**$ADJUSTEDPROBABILITY**
Sport-100	4334	0.29...	0.25...
Water Bottle	2866	0.19...	0.17...
Patch kit	2113	0.14...	0.13...

However, this type of query might affect performance in a production setting. This is because the query returns a set of all predictions from the algorithm, sorts these predictions, and returns the top 3.

The following example provides an alternative statement that returns the same results but executes significantly faster. This example replaces **TopCount** with the **Predict** function, which accepts a number of predictions as an argument. This example also uses the **$SUPPORT** keyword to directly retrieve the nested table column.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 3, $SUPPORT)
```

The results contain the top 3 predictions sorted by the support value. You can replace $SUPPORT with $PROBABILITY or $ADJUSTED_PROBABILITY to return predictions ranked by probability or adjusted probability. For more information, see **Predict (DMX)**.

See Also

Functions (DMX)

Mapping Functions to Query Types (DMX)

BottomCount (DMX)

TopPercent (DMX)

TopSum (DMX)

TopPercent

The **TopPercent** function returns, in order of decreasing rank, the top-most rows of a table whose cumulative total is at least a specified percentage.

Syntax

```
TopPercent(<table expression>, <rank expression>, <percent>)
```

Applies To

An expression that returns a table, such as a <table column reference>, or a function that returns a table.

Return Type

<table expression>

Remarks

The **TopPercent** function returns the top-most rows in decreasing order of rank based on the evaluated value of the <rank expression> argument for each row, such that the sum of the <rank expression> values is at least the given percentage that is specified by the <percent> argument. **TopPercent** returns the smallest number of elements possible while still meeting the specified percent value.

Examples

The following example creates a prediction query against the Association model that you build by using the Basic Data Mining Tutorial.

To understand how **TopPercent** works, it might be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)

FROM

     [Association]

NATURAL PREDICTION JOIN

SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note

In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851
Water Bottle	2866	0.192620472	0.175205052

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **TopPercent** function takes the results of this query and returns the rows with the greatest values that sum to the specified percentage.

```
SELECT
TopPercent
    (
    Predict ([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $SUPPORT,
    50)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **TopPercent** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **TopPercent** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $SUPPORT because support values are not fractional and therefore are easier to verify.

The third argument to the **TopPercent** function specifies the percentage, as a double. To get the rows for the top products that sum to 50 percent of the total support, you type 50.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.29...	0.25...
Water Bottle	2866	0.19...	0.17...
Patch kit	2113	0.14...	0.13...
Mountain Tire Tube	1992	0.133...	0.12...

Note This example is provided only to illustrate the usage of TopPercent. Depending on the size of your data set, this query might take a long time to run.

Warning
The MDX functions for TOPPERCENT and BOTTOMPERCENT can generate unexpected results when the values used to calculate the percentage include negative numbers. This behavior does not affect the DMX functions. For more information, see BOTTOMPERCENT.

See Also

Mapping Functions to Query Types (DMX)

Functions (DMX)

Mapping Functions to Query Types (DMX)

TopSum

Returns, in order of decreasing rank, the top-most rows of a table whose cumulative total is at least a specified value.

Syntax

```
TopSum(<table expression>, <rank expression>, <sum>)
```

Applies To

An expression that returns a table, such as a <table column reference>, or a function that returns a table.

Return Type

<table expression>

Remarks

The **TopSum** function returns the top-most rows in decreasing order of rank based on the evaluated value of the <rank expression> argument for each row, such that the sum of the <rank expression> values is at least the given total that is specified by the <sum> argument. **TopSum** returns the smallest number of elements possible while still meeting the specified sum value.

Examples

The following example creates a prediction query against the Association model that you build by using the Basic Data Mining Tutorial.

To understand how **TopPercent** works, it might be helpful to first execute a prediction query that returns only the nested table.

```
SELECT Predict ([Association].[v Assoc Seq Line Items],
INCLUDE_STATISTICS, 10)

FROM

     [Association]

NATURAL PREDICTION JOIN

SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

Note
In this example, the value supplied as input contains a single quotation mark, and therefore must be escaped by prefacing it with another single quotation mark. If you are not sure of the syntax for inserting an escape character, you can use the Prediction Query Builder to create the query. When you select the value from the dropdown list, the required escape character is inserted for you. For more information, see How to: Create a Singleton Query in the Data Mining Designer.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.291283016	0.252695851
Water Bottle	2866	0.192620472	0.175205052
Patch kit	2113	0.142012232	0.132389356
Mountain Tire Tube	1992	0.133879965	0.125304948
Mountain-200	1755	0.117951475	0.111260823

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Road Tire Tube	1588	0.106727603	0.101229538
Cycling Cap	1473	0.098998589	0.094256014
Fender Set - Mountain	1415	0.095100477	0.090718432
Mountain Bottle Cage	1367	0.091874454	0.087780332
Road Bottle Cage	1195	0.080314537	0.077173962

The **TopSum** function takes the results of this query and returns the rows with the greatest values that sum to the specified count.

```
SELECT
TopSum
    (
    Predict([Association].[v Assoc Seq Line
Items],INCLUDE_STATISTICS,10),
    $PROBABILITY,
    .5)
FROM
     [Association]
NATURAL PREDICTION JOIN
(SELECT (SELECT 'Women''s Mountain Shorts' as [Model]) AS [v Assoc Seq
Line Items]) AS t
```

The first argument to the **TopSum** function is the name of a table column. In this example, the nested table is returned by calling the **Predict** function and using the INCLUDE_STATISTICS argument.

The second argument to the **TopSum** function is the column in the nested table that you use to order the results. In this example, the INCLUDE_STATISTICS option returns the columns $SUPPORT, $PROBABILTY, and $ADJUSTED PROBABILITY. This example uses $PROBABILITY to return rows that sum to at least 50% probability.

The third argument to the **TopSum** function specifies the target sum, as a double. To get the rows for the top products that sum to 50 percent probability, you type .5.

Example results:

Model	$SUPPORT	$PROBABILITY	$ADJUSTEDPROBABILITY
Sport-100	4334	0.29…	0.25…
Water Bottle	2866	0.19…	0.17…
Patch kit	2113	0.14…	0.13…

Note This example is provided only to illustrate the usage of **TopSum**. Depending on the size of your data set, this query might take a long time to run.

See Also

Functions (DMX)

Mapping Functions to Query Types (DMX)

TopPercent (DMX)

Data Mining Extensions (DMX) Operator Reference

The Data Mining Extensions (DMX) language in Microsoft SQL Server Analysis Services supports arithmetic, assignment, comparison, logical, and unary operators. The following table lists the operators that DMX supports.

Operator	Description
Operators (DMX)	An arithmetic operator that adds two numbers together.
- (Subtract) (DMX)	An arithmetic operator that subtracts one number from another number.
* (Multiply) (DMX)	An arithmetic operator that multiplies one number by another number.
/ (Divide) (DMX)	An arithmetic operator that divides one number by another number.
< (Less Than) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is less than the value of the argument on the right; returns FALSE otherwise. If either

Operator	Description
	argument or both arguments evaluate to a null value, the operator returns a null value.
> (Greater Than) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is greater than the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
= (Equal To) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
<> (Not Equal To) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is not equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
<= (Less Than or Equal To) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is less than or equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.
>= (Greater Than or Equal To) (DMX)	A comparison operator. For arguments that evaluate to non-null values, returns TRUE if the value of the argument on the left is greater than or equal to the value of the argument on the right; returns FALSE otherwise. If either argument or both arguments evaluate to a null value, the operator returns a null value.

Operator	Description
AND (DMX)	A logical operator that performs a conjunction on two numeric expressions.
NOT (DMX)	A logical operator that performs a negation on a numeric expression.
OR (DMX)	A logical operator that performs a disjunction on two numeric expressions.
+ (Positive) (DMX)	A unary operator that returns the positive value of a numeric expression.
- (Negative) (DMX)	A unary operator that returns the negative value of a numeric expression.
// (Comment) (DMX)	Indicates a text string that Analysis Services should not execute. You can nest comments within a DMX statement, include them at the end of a line of code, or insert them on a separate line.
-- (Comment) (DMX)	Indicates a text string that Analysis Services should not execute. You can nest comments within a DMX statement, include them at the end of a line of code, or insert them on a separate line.
/*...*/ (Comment) (DMX)	Indicates a text string that Analysis Services should not execute. You can nest comments within a DMX statement, include them at the end of a line of code, or insert them on a separate line.

See Also

Data Mining Extensions (DMX) Function Reference

Data Mining Extensions (DMX) Reference

Data Mining Extensions (DMX) Statement Reference

Data Mining Extensions (DMX) Syntax Conventions

Data Mining Extensions (DMX) Syntax Elements

Operators (DMX)

+ (Add)

Performs an arithmetic operation that adds two numbers together.

Syntax

```
Numeric_Expression + Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Data Mining Extensions (DMX) expression that returns a numeric value.

Return Value

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

Remarks

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

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

- (Subtract)

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

Syntax

```
Numeric_Expression - Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Data Mining Extensions

	(DMX) expression that returns a numeric value.

Return Value

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

Remarks

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

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

* (Multiply)

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

Syntax

```
Numeric_Expression * Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Data Mining Extensions (DMX) expression that returns a numeric value.

Return Value

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

Remarks

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

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

/ (Divide)

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

Syntax

```
Dividend / Divisor
```

Parameters

Parameter	Description
Dividend	A valid Data Mining Extensions (DMX) expression that returns a numeric value.
Divisor	A valid DMX expression that returns a numeric value.

Return Value

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

Remarks

The value that this operator returns represents the quotient of the first expression divided by the second expression.

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

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

/ (Divide) (SSIS)

/ (Divide) (Transact-SQL)

< (Less Than)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is less than the value of another DMX expression.

Syntax

```
DMX_Expression < DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the first parameter has a value that is less than the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the first parameter has a value that is equal to or greater than the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

> (Greater Than)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is greater than the value of another DMX expression.

Syntax

```
DMX_Expression > DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the first parameter has a value that is greater than the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the first parameter has a value that is equal to or less than the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

= (Equal To)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is equal to the value of another DMX expression.

Syntax

```
DMX_Expression  = DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the value of the first parameter is equal to the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the value of the first parameter is not equal to the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

<> (Not Equal To)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is not equal to the value of another DMX expression.

Syntax

```
DMX_Expression <> DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the value of the first parameter is not equal to the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the value of the first parameter is equal to the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

<= (Less Than or Equal To)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is less than or equal to the value of another DMX expression.

Syntax

```
DMX_Expression <= DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the value of the first parameter is less than or equal to the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the value of the first parameter is greater than the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

>= (Greater Than or Equal To)

Performs a comparison operation that determines whether the value of one Data Mining Extensions (DMX) expression is greater than or equal to the value of another DMX expression.

Syntax

```
DMX_Expression >= DMX_Expression
```

Parameters

Parameter	Description
DMX_Expression	A valid DMX expression.

Return Value

A Boolean value that contains TRUE if both parameters are non-null and the value of the first parameter is greater than or equal to the value of the second parameter. The Boolean value contains FALSE if both parameters are non-null and the value of the first parameter is less than the value of the second parameter. The Boolean value contains a null value if either parameter or both parameters evaluate to a null value.

See Also

Operators (DMX)

Data Mining Extensions (DMX) Operator Reference

Operators (DMX)

AND

Performs a logical conjunction on two numeric expressions.

Syntax

Expression1 AND Expression2

Parameters

Parameter	Description
Expression1	A valid Data Mining Extensions (DMX) expression that returns a numeric value.
Expression2	A valid DMX expression that returns a numeric value.

Return Value

A Boolean value that returns TRUE if both parameters evaluate to TRUE; otherwise FALSE.

Remarks

Both parameters are treated as Boolean values (0 as FALSE; otherwise TRUE) before the operator performs the logical conjunction. The following table lists the values that are returned based on the various combinations of parameter values.

If Expression1 is	**If Expression2 is**	**Return value is**
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

See Also

Operators (DMX)

Logical Operators (DMX)

Operators (DMX)

NOT

A logical operator that performs a logical negation on a numeric expression.

Syntax

NOT Expression1

Parameters

Parameter	Description
Expression1	A valid DMX expression that returns a numeric value.

Return Value

A Boolean value that returns FALSE if the argument evaluates to TRUE; otherwise FALSE.

Remarks

The argument is treated as a Boolean value (0 as FALSE; otherwise TRUE) before the operator performs the logical negation. If Expression1 is TRUE, the operator returns FALSE. If Expression1 is FALSE, the operator returns TRUE. The following table illustrates how the logical conjunction is performed.

If Expression1 is	**Return value is**
TRUE	FALSE
FALSE	TRUE

See Also

Operators (DMX)

Logical Operators (DMX)

Operators (DMX)

OR

A logical operator that performs a logical disjunction on two numeric expressions.

Syntax

Expression1 OR Expression2

Parameters

Parameter	Description
Expression1	A valid Data Mining Extensions (DMX) expression that returns a numeric value.
Expression2	A valid DMX expression that returns a numeric value.

Return Value

A Boolean value that returns TRUE if either argument or both arguments evaluate to TRUE; otherwise FALSE.

Remarks

Both arguments are treated as Boolean values (0 as FALSE; otherwise TRUE) before the operator performs the logical disjunction. If either argument or both arguments evaluate to TRUE, the operator returns TRUE. If Expression1 evaluates to TRUE and Expression2 evaluates to FALSE, the operator returns TRUE.

The following table illustrates how the logical disjunction is performed.

If Expression1 is	**If Expression2 is**	**Return value is**
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

See Also

Operators (DMX)

Logical Operators (DMX)

Operators (DMX)

+ (Positive)

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

Syntax

```
+ Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Data Mining Extensions (DMX) expression that returns a numeric value.

Return Value

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

See Also

Unary Operators (DMX)
Operators (DMX)
Unary Operators

- (Negative)

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

Syntax

```
- Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Data Mining Extensions (DMX) expression that returns a numeric value.

Return Value

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

See Also

Unary Operators (DMX)

Operators (DMX)

Unary Operators

// (Comment)

Indicates a text string that Analysis Services should not execute. You can nest comments within a Data Mining Extensions (DMX) statement, include them at the end of a line of code, or insert them on a separate line.

Syntax

```
// Comment_Text
```

Parameters

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

Remarks

Use // for single-line comments only. Comments that are inserted by using // are delimited by the newline character.

There is no maximum length for comments.

For more information about how to use different kinds of comments in DMX, see Comments (DMX).

See Also

Operators (DMX)

-- (Comment) (DMX)

DMX Operator Reference

Operators (DMX)

-- (Comment)

Indicates a text string that Analysis Services should not execute. You can nest comments within a Data Mining Extensions (DMX) statement, include them at the end of a line of code, or insert them on a separate line.

Syntax

```
-- Comment_Text
```

Parameters

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

Remarks

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

There is no maximum length for comments.

For more information about how to use different kinds of comments in DMX, see Comments (DMX).

See Also

Operators (DMX)

// (Comment) (DMX)

DMX Operator Reference

Operators (DMX)

/*...*/ (Comment)

Indicates a text string that Analysis Services should not execute. The server does not evaluate the text between the comment characters /* and */. You can nest comments within a Data Mining Extensions (DMX) statement, include them at the end of a line of code, or insert them on a separate line.

Syntax

```
/* Comment_Text */
```

Parameters

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

Remarks

Multiple-line comments must be indicated by /* and */.

There is no maximum length for comments.

For more information about how to use different kinds of comments in DMX, see Comments (DMX).

See Also

Operators (DMX)

-- (Comment) (DMX)

DMX Operator Reference

Operators (DMX)

Data Mining Extensions (DMX) Syntax Conventions

The Data Mining Extensions (DMX) reference documentation in Microsoft SQL Server Analysis Services uses the following conventions to describe the DMX language.

Convention	Usage
bold	DMX keywords and text that must be typed exactly as shown.
italic	User-supplied arguments of DMX syntax.
| (vertical bar)	Used to separate syntax items within brackets or braces. You can choose only one of the items.
[ ] (brackets)	Contain optional syntax items. Do not type the brackets.
{ } (braces)	Contain required syntax items. Do not type the braces.
, ...	Indicates that the item before the comma can be repeated any number of times. The items are separated by commas.
<label> ::=	The name for a block of syntax. This convention is used to group and label sections of lengthy syntax or a unit of

Convention	Usage
	syntax that can be used in more than one location within a statement. Each location in which the block of syntax can be used is indicated with the label enclosed in chevrons, such as <label>.

See Also

Data Mining Extensions (DMX) Reference

DMX Tutorials (Analysis Services - Data Mining)

The following tutorials introduce you to the use of Data Mining Extensions (DMX) statements with data mining structures and models.

In this Section

Bike Buyer DMX Tutorial

In this tutorial, you will learn how to create, train, and explore mining models by using the DMX query language. You will then use these mining models to create predictions about whether a customer is likely to purchase a specific product.

Market Basket DMX Tutorial

In this tutorial, you will learn how to create a mining model that predicts which products tend to be purchased at the same time. This tutorial also demonstrates the use of nested tables in data mining.

Reference

Data Mining Extensions (DMX) Syntax Elements

Data Mining Extensions (DMX) Data Definition Statements

Data Mining Extensions (DMX) Data Manipulation Statements

Understanding the Select Statement (DMX)

Related Sections

Analysis Services Data Access Interfaces (Analysis Services - Multidimensional Data)

See Also

Prediction Queries (DMX)

Basic Data Mining Tutorial