

Teradata Vantage™ - NewSQL Engine Analytic Functions

Release 16.20

July 2019






Copyright and Trademarks

Copyright © 2017 - 2019 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

Product Safety

Safety type	Description
	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: teradata-books@lists.teradata.com

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Contents

Chapter 1: Introduction to Teradata Vantage NewSQL Engine Analytic Functions	4
Teradata Vantage NewSQL Engine Analytic Functions Overview	4
Usage Notes	5
AA 7.00 Usage Notes	7
Chapter 2: Teradata Vantage NewSQL Engine Analytic Functions	20
Antiselect	20
Attribution	23
DecisionForestPredict	31
DecisionTreePredict	40
GLMPredict	48
MovingAverage	60
NaiveBayesPredict	75
NaiveBayesTextClassifierPredict	81
NGramSplitter	85
nPath®	90
Pack	127
Sessionize	132
StringSimilarity	135
SVMSparsePredict	140
Unpack	144
Appendix A: Additional Information	153

Introduction to Teradata Vantage NewSQL Engine Analytic Functions

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

Teradata NewSQL Engine is a core of Teradata Vantage, based on our best-in-class Teradata Database processing capability. NewSQL refers to the ability to run advanced analytic functions beyond that of standard SQL.

Teradata Vantage NewSQL Engine Analytic Functions Overview

Teradata® NewSQL Engine (NewSQL Engine) analytic functions are specifically for analyzing data. Data can include clickstreams, financial transaction data, and user interaction data.

Function Name	Description
Antiselect	AntiSelect returns all columns <i>except</i> those specified.
Attribution	Calculates attributions with a wide range of distribution models. Often used in web-page analysis.
DecisionForestPredict	Uses the model file output by Teradata® Machine Learning Engine (ML Engine) DecisionForest function to analyze the input data and make predictions.
DecisionTreePredict	Uses the model file output by ML Engine DecisionTree function to analyze the input data and make predictions.
GLMPredict	Uses the model file output by ML Engine GLM function to analyze the input data and make predictions.
MovingAverage	Computes average values in a series.
NaiveBayesPredict	Uses the model file output by ML Engine Naive Bayes Classifier function to analyze the input data and make predictions.
NaiveBayesTextClassifierPredict	Uses the model file output by ML Engine NaiveBayesTextClassifierTrainer function to analyze the input data and make predictions.
NGramSplitter	Tokenizes (splits) an input stream and emits <i>n</i> multigrams, based on specified delimiter and reset parameters. Useful for sentiment analysis, topic identification, and document classification.
nPath@	Performs regular pattern matching over a sequence of rows from one or more inputs.

Function Name	Description
Pack	Compresses data in multiple columns into a single packed data column.
Sessionize	Maps each click in a clickstream to a unique session identifier.
StringSimilarity	Calculates the similarity between two strings, using the specified comparison method.
SVMSparsePredict	Uses the model file output by ML Engine SVMSparse function to analyze the input data and make predictions.
Unpack	Expands data from a single packed column to multiple unpacked columns.

NewSQL Engine analytic functions have corresponding ML Engine functions. The syntax of corresponding functions may differ, but given the same inputs and syntax element values, they produce the same results (with the minor exceptions noted in specific functions).

Note:

Most NewSQL Engine analytic functions do not support column ranges, as many of their corresponding ML Engine functions do.

To execute ML Engine functions on Teradata Vantage™, contact your Teradata Support representative.

More Information

Topic	Reference
ML Engine functions	<i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003
Aster Analytics functions	<i>Teradata Aster® Analytics Foundation User Guide</i> , B700-1022
Installing model files output by ML Engine functions on NewSQL	<i>Teradata Vantage™ User Guide</i> , B700-4002

Usage Notes

These usage notes apply to every function in this document.

Function Syntax Descriptions

SELECT Statement Clauses

The function syntax descriptions in this document are SQL SELECT statements. For simplicity, the descriptions do not show every possible SELECT statement clause. However, you can use any valid

SELECT statement clauses. For information about SELECT statement options, see *Teradata Vantage™ SQL Data Manipulation Language*, B035-1146.

Many examples in this document use ORDER BY clauses that the function syntax descriptions do not show.

Function Syntax Element Order

Function syntax elements must appear after the USING clause, but they need not appear in the order shown in the function syntax description.

Many examples in this document do not specify their syntax elements in the order shown in the function syntax description.

Terminology

This document uses the following terms.

Term	Description
Path	An ordered, start-to-finish series of actions, for example, page views, for which sequences and sub-sequences can be created.
Sequence	A sequence is the path prefixed with a caret (^), which indicates the start of a path. For example, if a user visited page a, page b, and page c, in that order, the session sequence is ^,a,b,c.
Subsequence	For a given sequence of actions, a sub-sequence is one possible subset of the steps that begins with the initial action. For example, the path a,b,creates three subsequences: ^,a; ^,a,b; and ^,a,b,c.

Functions Ignore Disallowed Syntax Elements

If you call a function with a syntax element that is disallowed, the function ignores the syntax element, without issuing an error message.

Unexpected Results with Non-LATIN Characters

If a function description specifies "CHARACTER SET must be LATIN," all string syntax element values must have only Latin characters; otherwise, the function might output unexpected results.

Displaying Online Help for NewSQL Engine Analytic Functions

Online help is available for each NewSQL Engine analytic function.


- For information about a function, type:
`HELP 'function_name'`
 For example:
`HELP 'SQL NPATH'`

BC/BCE Timestamps

NewSQL Engine functions do not support Before the Common Era (BCE) timestamps. BCE is an alternative to Before Christ (BC). These are examples of BC/BCE timestamps:

```
4713-01-01 11:07:11-07:52:58 BC
4713-01-01 11:07:11 BC
```

Workload Management Configuration for NewSQL Engine Analytic Functions

NewSQL Engine analytic functions can be memory- and compute-intensive and impact other workloads, depending on function parameters and input table sizes. To learn to use Workload Management throttles to limit concurrency and memory, see *Workload Management Configuration for NewSQL Engine Analytic Functions*, which you can download from the download tab  in the left pane.

Workload Management Configuration for NewSQL Engine Analytic Functions is also available at <https://downloads.teradata.com/>.

AA 7.00 Usage Notes

These usage notes apply only if you use model tables created using a supported version of Aster Analytics on Aster Database as input to these functions:

- DecisionTreePredict
- DecisionForestPredict
- GLMPredict
- NaiveBayesPredict
- NaiveBayesTextClassifierPredict
- SVMSParsePredict

AA 7.00 Limitations

- The minimum supported version of Aster Analytics is AA 7.00.

Models created using an earlier version of Aster Analytics must be recreated after upgrading to a supported version of Aster Analytics.

AA 7.00 Model Tables

These notes apply only to model tables output by AA 7.00 functions.

- BLOB and CLOB data types are not supported.
- Aster data type BYTEA corresponds to Teradata Database data type VARBYTE.
- The maximum size of a VARBYTE or VARCHAR column is 64000.

- You can load a table created on Aster Database to Teradata Database using either the `load_to_teradata` command or Open Database Connectivity (ODBC).

Analytic Functions on Teradata Database and Aster Database

The following table summarizes the differences between analytic functions on Teradata Database and Aster Database.

Teradata Database Analytic Function	Aster Database Analytic Function
PARTITION BY clause lets you specify a column by its position, an integer. PARTITION BY 1 partitions rows by column 1.	PARTITION BY clause accepts only column names. PARTITION BY 1 causes the function to process all rows on a single worker node.
For table operator output, an alias is required.	For function output, an alias is optional.
To specify function syntax elements, you must use a USING clause.	Function syntax does not include USING clause.
Function syntax elements do not support column ranges.	Function syntax elements support column ranges.

Loading Aster Tables to Teradata Database Using `load_to_teradata`

For `load_to_teradata` instructions, see *Teradata Aster® Database User Guide* and the following usage notes.

`load_to_teradata` Usage Notes

- If a table column name includes a keyword, enclose the name in double quotation marks and alias it.
- In SELECT statements, enclose every camel-case table column name in double quotation marks.

This example shows both aliased columns and camel-case column names:

```
SELECT * FROM load_to_teradata (
  ON (
    SELECT "class" AS class_col,
           "variable" AS variable_col,
           "type" AS type_col,
           category,
           cnt,
           "sum" AS sum_col,
           "sumSq",
           "totalCnt"
    FROM aster_nb_modelSC
  )
)
```



```
tdpid ('sdt12432.labs.teradata.com')
username ('sample_user')
password ('sample_user')
target_table ('td_nb_modelSC')
);
```

- Cast every REAL column to DOUBLE PRECISION.

For example:

```
SELECT * FROM load_to_teradata (
  ON (
    SELECT attribute,
           predictor,
           category,
           CAST (estimate AS DOUBLE PRECISION) AS estimate,
           CAST (std_err AS DOUBLE PRECISION) AS std_err,
           CAST (z_score AS DOUBLE PRECISION) AS z_score,
           CAST (p_value AS DOUBLE PRECISION) AS p_value,
           significance,
           "family"
    FROM glm_housing_model
  )
  tdpid ('sdt12432.labs.teradata.com')
  username ('sample_user')
  password ('sample_user')
  target_table ('glm_housing_model')
);
```

- If a model table column name contains Teradata Database reserved keywords or special characters — characters other than letters, digits, or underscore (_)—enclose it in double quotation marks.

This rule applies to the following model column names:

AA 7.00 Function	Model Column Name
Single_Tree_Drive	node_gini(p) node_entropy(p) node_chisq_pv(p) split_gini(p) split_entropy(p) split_chisq_pv(p)
NaiveBayesReduce	class variable type sum sumSq

AA 7.00 Function	Model Column Name
	totalCnt

For example:

```
CREATE SET TABLE NBUSER.td_glass_modelPD1,
  FALLBACK,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT,
  DEFAULT MERGEBLOCKRATIO,
  MAP = TD_MAP1 (
    node_id BIGINT,
    node_size BIGINT,
    "node_gini(p)" FLOAT,
    node_entropy FLOAT,
    node_chisq_pv FLOAT,
    node_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    node_majorvotes BIGINT,
    split_value FLOAT,
    "split_gini(p)" FLOAT,
    split_entropy FLOAT,
    split_chisq_pv FLOAT,
    left_id BIGINT,
    left_size BIGINT,
    left_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    left_majorvotes BIGINT,
    right_id BIGINT,
    right_size BIGINT,
    right_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    right_majorvotes BIGINT,
    left_bucket VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    right_bucket VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    left_label_problist VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    right_label_problist VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    prob_label_order VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
    attribute VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
```

```
)  
PRIMARY INDEX (node_id);
```

Related Information:

[Loading Aster Tables to Teradata Database Using ODBC](#)

Loading Aster Tables to Teradata Database Using ODBC

The ODBC instructions follow. To follow them, you must have an account on <https://downloads.teradata.com>.

1. [Install Teradata Parallel Transporter Base.](#)
2. [Set up the Aster driver on the client machine.](#)
3. If the table does not exist on Aster Database, create and populate it there.
4. On Teradata Database, do the following:
 - a. If the user who is to own the table does not exist, create it.
 - b. [Write the tpt script.](#)
 - c. [Write the JobVariablesFile.](#)
 - d. [Use the tbuild command to run the tpt script.](#)

Related Information:

[Loading Aster Tables to Teradata Database Using load_to_teradata](#)
[Example: Loading Aster Table to Teradata Database Using ODBC](#)

Installing Teradata Parallel Transporter Base

1. Go to <https://access.teradata.com>.
2. Log in.
3. Download the package TTU 16.20.04.00 Windows - Base.
 Downloading the packages takes approximately 30-40 minutes.
4. On your client machine, go to the folder where the package was downloaded and unzip it.
5. Go to TeradataToolsAndUtilitiesBase\Windows and run TTU.exe.
6. Install Teradata Parallel Transporter Base.

Setting Up the Aster Driver on the Client

1. Go to <https://access.teradata.com>.
2. Log in.
3. Download AsterClients__windows_x8664.version.zip, where *version* is the version of Aster Analytics on your client machine; for example:
 AsterClients__windows_x8664.06.20.00.00.zip
 Downloading the packages takes several minutes.

4. On your client machine, go to the folder where the package was downloaded and unzip it.
5. Go to the subfolder `\stage\home\beehive\clients-win nn` , where nn is 32, 64, or 86, depending on your Windows machine. For example:
`\stage\home\beehive\clients-win64`
6. Install `nClusterODBCInstaller_ xnn` .
If the installer requests a dependency package, install it from the web.
7. Open ODBC Data Sources (nn -bit).
8. On the **System DSN** tab, click **Add**.
If Aster ODBC driver installation succeeded, the window **Aster ODBC Driver** appears.
9. In the window **Aster ODBC Driver**, click **Finish**.
10. In the DSN Setup form that appears, enter the following values and click **OK**:

Field	Value
Data Source	Name of data source to use in tpt script
Server	IP address of Aster queen
Port	2406
Database	Aster Database
Username	User name
Password	User password
MaxLenVarchar	Default length of unbounded VARCHAR data item

11. Click **OK**.

Writing the tpt Script

- Write the tpt script by substituting values for variables in the following script:

```

DEFINE JOB PRODUCT_SOURCE_LOAD

DESCRIPTION 'LOAD PRODUCT DEFINITION TABLE'
(
  DEFINE SCHEMA PRODUCT_SOURCE_SCHEMA
  DESCRIPTION 'PRODUCT INFORMATION SCHEMA'
  (
    TD_compatible_table_definition

    STEP STEP_CREATE_DDL
    (
      APPLY
      ('DROP TABLE ' || @TargetTable || ' ;'),

```

```

        ('CREATE MULTISET TABLE ' ||
@TargetTable|| '(TD_compatible_table_definition;')
        TO OPERATOR ($DDL() [1]);
    );
Step Insert_Tables
(
    APPLY
    ('Ins ' ||@TargetTable|| '(
        : "column_name_1"
        , : "column_name_2"
        [ ..., : "column_name_k" ]
    );
)
TO OPERATOR ($LOAD()[1])

SELECT * FROM OPERATOR ($ODBC(PRODUCT_SOURCE_SCHEMA)[1]);
);

```

Writing the JobVariablesFile

- Write the JobVariablesFile by substituting values for variables in the following script:

```

DDLTDpId           = 'td_host_name_or_ip'
,DDLUserName       = 'td_user'
,DDLUserPassword   = 'td_user_password'
,DDLErrorList      = ['3807']
,DDLPrivateLogName = 'DDL001S1'
,TargetTable       = 'td_table_name'
,ODBCPrivateLogName = 'ODB039P1'
,ODBCDSNName       = 'Data_Source_Name_specified_in_DSN_Setup_form'
,TruncateData      = {'Y' | 'N'}
,ODBCUserName      = 'aster_user'
,ODBCUserPassword  = 'aster_user_password'
,LOADPrivateLogName = 'ODB039C1'
,LOADTDPID         = 'td_host_name_or_ip'
,LOADUserName      = 'td_user'
,LOADUserPassword  = 'td_user_password'
,SelectStmt        = 'SELECT * FROM aster_table_name;'
,LOADTargetTable   = 'td_table_name'

```

Note:

For TruncateData, 'Y' trims unused space. The default is 'N'. Specify 'Y' when the MaxLenVarchar field of the DSN Setup form (in [Setting Up the Aster Driver on the Client](#)) exceeds the maximum VARCHAR length specified in *TD_compatible_table_definition* in the tpt script; otherwise, ODBC cannot load the script.

Running the tpt Script

You are on Teradata Database, where a folder contains the tpt script and JobVariablesFile that you wrote.

1. Open the command prompt.
2. Go to the folder where the tpt script and JobVariablesFile are.
3. Run the tpt script with this command:

```
tbuild -f tptfile -v JobVariablesFile -j jobid
```

where *tptfile* and *JobVariablesFile* are the names of the tpt script and JobVariablesFile that you wrote and *jobid* is the name you are giving to this tbuild job.

Example: Loading Aster Table to Teradata Database Using ODBC

This example shows the code for the following:

- Creating and populating a table on Aster Database
- Creating a Teradata Database user to own the table
- A tpt script
- A JobVariablesFile
- Running the tpt script

Code for Creating and Populating a Table on Aster Database

This code creates and populates a training table, aster_nb_trainerSC, and then uses the training table and Naive Bayes Classifier function to create a model table, aster_nb_modelSC.

```
/* Create training table */

DROP TABLE IF EXISTS aster_nb_trainerSC;

CREATE TABLE aster_nb_trainerSC (
  id      INT,
  year    INT,
  color   VARCHAR (100),
  type    VARCHAR (100),
  origin  VARCHAR (100),
```

```

    stolen VARCHAR (100),
    PARTITION KEY (id)
);

/* Populate training table */

INSERT INTO aster_nb_trainerSC VALUES
  (1,3,'red','sports','domestic','Yes'),
  (2,9,'red','sports','domestic','No'),
  (3,1,'red','sports','domestic','Yes'),
  (4,8,'yellow','sports','domestic','No'),
  (5,2,'yellow','sports','imported','Yes');

/* Create model table from training table */

DROP TABLE IF EXISTS aster_nb_modelSC;

CREATE TABLE aster_nb_modelSC distribute by hash(class_nb) AS (
  SELECT * FROM naiveBayesReduce (
    ON (
      SELECT * FROM naiveBayesMap (
        ON aster_nb_trainerSC
        Response ('stolen')
        NumericInputs ('year')
        CategoricalInputs ('color','origin','type')
      )
    ) PARTITION BY class_nb
  )
);

```

Code for Creating a Teradata Database User

```

CREATE USER sample_user AS
  PASSWORD = sample_user
  PERM = 10e6*(HASHAMP()+1);

GRANT ALL ON dbc TO sample_user;

```

tpt Script, astermodel.tpt

```

DEFINE JOB PRODUCT_SOURCE_LOAD
DESCRIPTION 'LOAD PRODUCT DEFINITION TABLE'
(
  DEFINE SCHEMA PRODUCT_SOURCE_SCHEMA
  DESCRIPTION 'PRODUCT INFORMATION SCHEMA'
  (

```

```

class_nb VARCHAR(128),
variable_nb VARCHAR(128),
type_nb VARCHAR(128),
category VARCHAR(32),
cnt BIGINT,
sum_nb FLOAT,
sum_sq FLOAT,
total_cnt BIGINT
);

STEP STEP_CREATE_DDL
(
  APPLY
    ('DROP TABLE '||@TargetTable||';'),
    ('CREATE MULTISET TABLE '||@TargetTable||'(
      class_nb VARCHAR(128),
      variable_nb VARCHAR(128),
      type_nb VARCHAR(128),
      category VARCHAR(32),
      cnt BIGINT,
      sum_nb FLOAT,
      sum_sq FLOAT,
      total_cnt BIGINT) NO PRIMARY INDEX;
    ')
  TO OPERATOR ($DDL()[1]);
);
Step Insert Tables
(
  APPLY
    ('Ins '||@TargetTable||'(
      :class_nb"
      ,:"variable_nb"
      ,:"type_nb"
      ,:"category"
      ,:"cnt"
      ,:"sum_nb"
      ,:"sum_sq"
      ,:"total_cnt");'
    )
  TO OPERATOR ($LOAD()[1])

  SELECT * FROM OPERATOR ($ODBC(PRODUCT_SOURCE_SCHEMA)[1]);
);
);

```


JobVariablesFile, attr.txt

```

DLTDpid = 'td_host_name_or_ip'
,DDLUserName = 'alice'
,DDLUserPassword = 'alice'
,DDLErrorList = '[3807]'
,DDLPrivateLogName = 'DDL001S1'
,TargetTable = 'td_nb_modelsc'
,ODBCPrivateLogName = 'ODBC039P1'
,ODBCDSNName = 'shruti'
,TruncateData = 'Y'
,ODBCUserName = 'db_superuser'
,ODBCUserPassword = 'db_superuser'
,LOADPrivateLogName = 'ODBC039P1'
,LOADTDPID = 'td_host_name_or_ip'
,LOADUserName = 'alice'
,LOADUserPassword = 'alice'
,SelectStmt = 'SELECT * FROM aster_nb_modelsc;'
,LOADTargetTable = 'td_nb_modelsc'

```

Command for Running the tpt Script

```
tbuild -f aster_model.tpt -v attr.txt -j urr1
```

Aster Model Table Schemas**Forest_Predict Model Table Schema**

```

CREATE FACT TABLE public.aster_fp_admissions_clsmodel
(
  worker_ip VARCHAR,
  task_index INTEGER,
  tree_num INTEGER,
  tree VARCHAR
)
DISTRIBUTE BY HASH (task_index)
STORAGE ROW;

```

GLMPredict Model Table Schema

```

CREATE ANALYTIC FACT TABLE public.glm_housing_model
(
  attribute INTEGER,
  predictor VARCHAR(1024),
  category VARCHAR(1024),

```

```

estimate DOUBLE PRECISION,
std_err DOUBLE PRECISION,
{ t_score | z_score } DOUBLE PRECISION,
p_value DOUBLE PRECISION,
significance VARCHAR(50),
family VARCHAR(20)
)
DISTRIBUTE BY HASH (attribute)
STORAGE ROW;

```

The model table has the column `t_score` if created with Family ('GAUSSIAN'), otherwise it has the column `z_score`.

NaiveBayesPredict Model Table Schema

```

CREATE ANALYTIC FACT TABLE public.aster_nb_modelsc
(
  class_nb VARCHAR(128),
  variable_nb VARCHAR(128),
  type_nb VARCHAR(128),
  category VARCHAR(32),
  cnt BIGINT,
  sum_nb DOUBLE PRECISION,
  sum_sq DOUBLE PRECISION,
  total_cnt BIGINT
)
DISTRIBUTE BY HASH (class_nb)
STORAGE ROW;

```

NaiveBayesTextClassifierPredict Model Table Schema

```

CREATE DIMENSION TABLE public.nbtcp_spam_multinomialmodel
(
  token VARCHAR,
  category VARCHAR,
  prob DOUBLE PRECISION
)
DISTRIBUTE BY REPLICATION
STORAGE ROW;

```

Single_Tree_Predict Model Table Schema

```

CREATE DIMENSION TABLE public.glass_model
(
  node_id BIGINT,
  node_size BIGINT,

```

```

node_gini_p DOUBLE PRECISION,
node_entropy DOUBLE PRECISION,
node_chisq_pv DOUBLE PRECISION,
node_label VARCHAR(512),
node_majorvotes BIGINT,
split_value DOUBLE PRECISION,
split_gini_p DOUBLE PRECISION,
split_entropy DOUBLE PRECISION,
split_chisq_pv DOUBLE PRECISION,
left_id BIGINT,
left_size BIGINT,
left_label VARCHAR(512),
left_majorvotes BIGINT,
right_id BIGINT,
right_size BIGINT,
right_label VARCHAR(512),
right_majorvotes BIGINT,
left_bucket VARCHAR(512),
right_bucket VARCHAR(512),
attribute VARCHAR(512)
)
DISTRIBUTE BY REPLICATION
STORAGE ROW;

```

SparseSVMPredict Model Table Schema

```

CREATE FACT TABLE public.aster_svm_iris_model_default
(
  classid INTEGER,
  weights BYTEA
)
DISTRIBUTE BY HASH (classid)
STORAGE ROW;

```

Teradata Vantage NewSQL Engine Analytic Functions

Antiselect

Antiselect returns all columns *except* those specified in the Exclude syntax element.

Note:

This function requires the UTF8 client character set.

Antiselect Syntax

```
SELECT * FROM Antiselect (
  ON { table | view | (query) }
  USING
  Exclude ({ 'exclude_column' | exclude_column_range }[,...])
) AS alias;
```

Antiselect Syntax Elements

Exclude

Specify the names of the input table columns to exclude from the output table. Column names must be valid object names, which are defined in *Teradata Vantage™ SQL Fundamentals*, B035-1141.

The *exclude_column* is a column name. This is the syntax of *exclude_column_range*:

```
'start_column:end_column' [, '-exclude_in-range_column' ]
```

The range includes its endpoints.

The *start_column* and *end_column* can be:

- Column names (for example, 'column1:column2')

Column names must contain only letters in the English alphabet, digits, and special characters. If a column name includes any special characters, surround the column name with double quotation marks. For example, if the column name is a*b, specify it as "a*b". A column name cannot contain a double quotation mark.

- Nonnegative integers that represent the indexes of columns in the table (for example, '[0:4]')

The first column has index 0; therefore, '[0:4]' specifies the first five columns in the table.

- Empty. For example:

- '[:4]' specifies all columns up to and including the column with index 4.
- '[4:]' specifies the column with index 4 and all columns after it.
- '[:]' specifies all columns in the table.

The *exclude_in-range_column* is a column in the specified range, represented by either its name or its index (for example, '[0:99]', '-[50]', '-column10' specifies the columns with indexes 0 through 99, except the column with index 50 and column10).

Column ranges cannot overlap, and cannot include any specified *exclude_column*.

Antiselect Input

The input table can have any schema.

Antiselect Output

The output table has all input table columns except those specified by the Exclude syntax element.

Antiselect Examples

Antiselect Example: No Column Ranges

Input

The input table, *antiselect_test*, is a sample set of sales data containing 13 columns.

antiselect_test

sno	id	orderdate	priority	qty	sales	disct	dmode	custname	province	region	cust
1	3	2010-10-13 00:00:00	Low	6	261. 54	0.04	Regular Air	Muhammed MacIntyre	Nunavut	Nunavut	Sma Busi
49	293	2012-10-01 00:00:00	High	49	10123	0.07	Delivery Truck	Barry French	Nunavut	Nunavut	Cons
50	293	2012-10-01 00:00:00	High	27	244. 57	0.01	Regular Air	Barry French	Nunavut	Nunavut	Cons
80	483	2011-07-10 00:00:00	High	30	4965. 76	0.08	Regular Air	Clay Rozendal	Nunavut	Nunavut	Corp
85	515	2010-08-28 00:00:00	Not specified	19	394. 27	0.08	Regular Air	Carlos Soltero	Nunavut	Nunavut	Cons
86	515	2010-08-28 00:00:00	Not specified	21	146. 69	0.05	Regular Air	Carlos Soltero	Nunavut	Nunavut	Cons
97	613	2011-06-17 00:00:00	High	12	93.54	0.03	Regular Air	Carl Jackson	Nunavut	Nunavut	Corp

SQL Call

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', 'orderdate', 'disct', 'province', 'custsegment')
) AS dt ORDER BY 1, 4;
```

Output

sno	priority	qty	sales	dmode	custname	region	prodcats
1	Low	6	2. 61540000000000E 002	Regular Air	Muhammed MacIntyre	Nunavut	Office Supplies
49	High	49	1. 01230000000000E 004	Delivery Truck	Barry French	Nunavut	Office Supplies
50	High	27	2. 44570000000000E 002	Regular Air	Barry French	Nunavut	Office Supplies
80	High	30	4. 96576000000000E 003	Regular Air	Clay Rozendal	Nunavut	Technology
85	Not specified	19	3. 94270000000000E 002	Regular Air	Carlos Soltero	Nunavut	Office Supplies
86	Not specified	21	1. 46690000000000E 002	Regular Air	Carlos Soltero	Nunavut	Furniture
97	High	12	9. 35400000000000E 001	Regular Air	Carl Jackson	Nunavut	Office Supplies

Antiselect Example: Column Range**Input**

The input table is antiselect_test, as in [Antiselect Example: No Column Ranges](#).

SQL Call

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
```

```
Exclude ('id', '[2:3]', 'custname:prodcat')
) AS dt ORDER BY 1, 4;
```

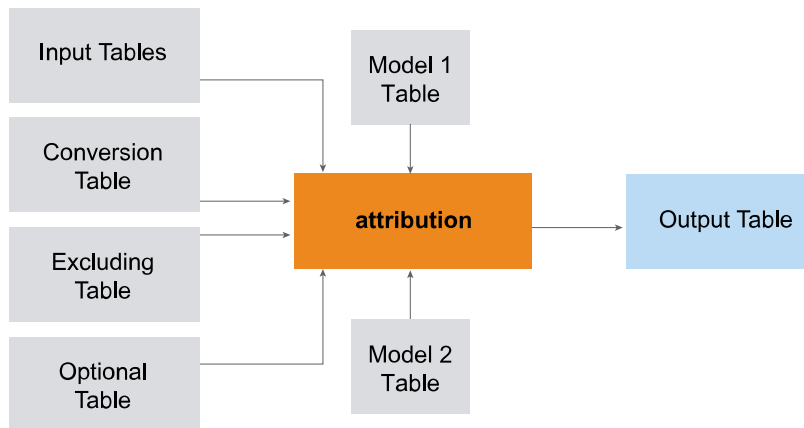
Output

sno	qty	sales	disct	dmode
1	6	2.61540000000000E 002	0.04	Regular Air
49	49	1.01230000000000E 004	0.07	Delivery Truck
50	27	2.44570000000000E 002	0.01	Regular Air
80	30	4.96576000000000E 003	0.08	Regular Air
85	19	3.94270000000000E 002	0.08	Regular Air
86	21	1.46690000000000E 002	0.05	Regular Air
97	12	9.35400000000000E 001	0.03	Regular Air

Attribution

The Attribution function is used in web page analysis, where it lets companies assign weights to pages before certain events, such as buying a product.

The function takes data and parameters from multiple tables and outputs attributions.



Note:

- ML Engine function Attribution_MLE has two versions:
 - Multiple-input: Accepts one or more input tables and gets many parameters from other dimension tables.
 - Single-input: Accepts only one input table and gets all parameters from syntax elements.

NewSQL Engine Attribution function corresponds to the multiple-input version. Unlike Attribution_MLE, Attribution does not support Unicode.

- A query that runs longer than 3 seconds before displaying output indicates that syntax elements supplied to the function are incorrect.

Attribution Syntax

```
SELECT * FROM Attribution (
  ON { table | view | (query) } [ AS InputTable1 ]
  PARTITION BY user_id
  ORDER BY times_column
  [ ON { table | view | (query) } [ AS InputTable2 ]
  PARTITION BY user_id
  ORDER BY time_column [,...] ]
  ON conversion_event_table AS ConversionEventTable DIMENSION
  [ ON excluding_event_table AS ExcludedEventTable DIMENSION ]
  [ ON optional_event_table AS OptionalEventTable DIMENSION ]
  ON model1_table AS FirstModel DIMENSION
  [ ON model2_table AS SecondModel DIMENSION ]
  USING
  EventColumn ('event_column')
  TimeColumn ('time_column')
  WindowSize ({'rows:K' | 'seconds:K' | 'rows:K&seconds:K2'})
) AS alias ORDER BY user_id,time_stamp;
```

Attribution Syntax Elements

EventColumn

Specify the name of the input column that contains the clickstream events.

TimeColumn

Specify the name of the input column that contains the timestamps of the clickstream events.

WindowSize

Specify how to determine the maximum window size for the attribution calculation:

Option	Description
rows : K	Assign attributions to at most K events before conversion event, excluding events of types specified in ExcludedEventTable.
seconds : K	Assign attributions only to rows not more than K seconds before conversion event.
rows : K & seconds : $K2$	Apply both constraints and comply with stricter one.

Attribution Input

Required

Table	Description
Input tables (maximum of five)	Contain clickstream data for computing attributions.
ConversionEventTable	Contains conversion events.
FirstModel	Defines type and distributions of first model.

Optional

Table	Description
ExcludedEventTable	Contains events to exclude from attribution.
OptionalEventTable	Contains optional events.
SecondModel	Defines type and distributions of second model.

Input Table Schema

Column	Data Type	Description
<i>userid_column</i>	INTEGER or VARCHAR	User identifier.
<i>event_column</i>	INTEGER or VARCHAR	Event from clickstream.
<i>time_column</i>	INTEGER, SMALLINT, BIGINT, TIMESTAMP, or TIME	Event timestamp.

ConversionEventTable Schema

Column	Data Type	Description
<i>conversion_event</i>	VARCHAR	[Column appears once for each conversion event.] Conversion event value (string or integer).

FirstModel and SecondModel Schema

Column	Data Type	Description
id	INTEGER	Row identifier. Rows are numbered 0, 1, 2, and so on.
model	VARCHAR	Row 0: Model type. Row 1, ..., <i>n</i> : Distribution model definition. SIMPLE model: Model table has single row that specifies model type and parameters. Other model types: <i>n</i> is number of rows or events included in model. For model type and specification definitions, see Model Specification .

ExcludedEventTable Schema

Column	Data Type	Description
<i>excluding_event</i>	VARCHAR	[Column appears once for each excluded event.] Excluded event (string or integer). Cannot be a conversion event.

OptionalEventTable Schema

Column	Data Type	Description
<i>optional_event</i>	VARCHAR	[Column appears once for each optional event.] Optional event (string or integer). Cannot be a conversion or excluded event. Function attributes conversion event to optional event only if it cannot attribute it to regular event.

Model Specification**Model Types and Specification Definitions**

Row 0: Model Type	Row 1, ..., <i>n</i> : Distribution Model Specification	Additional Information
SIMPLE	<i>MODEL:</i> <i>PARAMETERS</i>	Distribution model for all events. For MODEL and PARAMETER definitions, see following table.
EVENT_ REGULAR	<i>EVENT:WEIGHT:</i> <i>MODEL:</i> <i>PARAMETERS</i>	Distribution model for a regular event. EVENT cannot be a conversion, excluded, or optional event. For MODEL and PARAMETER definitions, see following table. Sum of WEIGHT values must be 1.0. For example, suppose that model table has these specifications: email:0.19:LAST_CLICK:NA impression:0.81:UNIFORM:NA Within WindowSize of a conversion event, 19% of conversion event is attributed to last email event and 81% is attributed uniformly to all impression events.

Row 0: Model Type	Row 1, ..., n: Distribution Model Specification	Additional Information
EVENT_OPTIONAL	<i>EVENT:WEIGHT:</i> <i>MODEL:</i> <i>PARAMETERS</i>	Distribution model for an optional event. EVENT must be in optional event table. For MODEL and PARAMETER definitions, see following table. Sum of WEIGHT values must be 1.0.
SEGMENT_ROWS	<i>K_i:WEIGHT:</i> <i>MODEL:</i> <i>PARAMETERS</i>	Distribution model by row. Sum of K_i values must be value K specified by 'rows: K ' in WindowSize syntax element. Function considers rows from most to least recent. For example, suppose that function call has these syntax elements: WindowSize ('rows:10') Model1 ('SEGMENT_ROWS', '3:0.5:UNIFORM:NA', '4:0.3:LAST_CLICK:NA', '3:0.2:FIRST_CLICK:NA') Attribution for a conversion event is divided among attributable events in 10 rows immediately preceding conversion event. If conversion event is in row 11, first model specification applies to rows 10, 9, and 8; second applies to rows 7, 6, 5, and 4; and third applies to rows 3, 2, and 1. Half attribution (5/10) is uniformly divided among rows 10, 9, and 8; 3/10 to last click in rows 7, 6, 5, and 4 (that is, in row 7), and 2/10 to first click in rows 3, 2, and 1 (that is, in row 1).
SEGMENT_SECONDS	<i>K_i:WEIGHT:</i> <i>MODEL:</i> <i>PARAMETERS</i>	Distribution model by time in seconds. Sum of K_i values must be value K specified by 'seconds: K ' in WindowSize syntax element. Function considers rows from most to least recent. For example, suppose that function call has these syntax elements: WindowSize ('seconds:20') Model1 ('SEGMENT_SECONDS', '6:0.5:UNIFORM:NA', '8:0.3:LAST_CLICK:NA', '6:0.2:FIRST_CLICK:NA') Attribution for a conversion event is divided among attributable events in 20 seconds immediately preceding conversion event. If conversion event is at second 21, first model specification applies to seconds 20-15 (counting backward); second applies to seconds 14-7; and third applies to seconds 6-1. Half attribution (5/10) is uniformly divided among seconds 20-15; 3/10 to last click in seconds 14-7, and 2/10 to first click in seconds 6-1.

MODEL Values and Corresponding PARAMETER Values

MODEL values are case-sensitive. Attributable events are those whose types are not specified in excluding events table.

MODEL	Description	PARAMETERS
'LAST_CLICK'	Conversion event is attributed entirely to most recent attributable event.	'NA'

MODEL	Description	PARAMETERS
'FIRST_CLICK'	Conversion event is attributed entirely to first attributable event.	'NA'
'UNIFORM'	Conversion event is attributed uniformly to preceding attributable events.	'NA'
'EXPONENTIAL'	Conversion event is attributed exponentially to preceding attributable events (the more recent the event, the higher the attribution).	' <i>alpha,type</i> ' where <i>alpha</i> is a decay factor in range (0, 1) and <i>type</i> is ROW, MILLISECOND, SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR. When <i>alpha</i> is in range (0, 1), sum of series $w_i = (1-\alpha) \cdot \alpha^i$ is 1. Function uses w_i as exponential weights.
'WEIGHTED'	Conversion event is attributed to preceding attributable events with weights specified by PARAMETERS. SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)	You can specify any number of weights. If there are more attributable events than weights, extra (least recent) events are assigned zero weight. If there are more weights than attributable events, then function renormalizes weights.

Allowed FirstModel/SecondModel Combinations

FirstModel Type	SecondModel Type
SIMPLE	Not allowed
EVENT_REGULAR	
EVENT_REGULAR	EVENT_OPTIONAL (when you specify optional events table)
SEGMENT_ROWS	SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)
SEGMENT_ROWS	
SEGMENT_SECONDS	Not allowed

Attribution Output

Attribution Output Table Schema

Column	Data Type	Description
user_id	INTEGER or VARCHAR	User identifier from input table.
event	VARCHAR	Clickstream event from input table.
time_stamp	TIMESTAMP	Event timestamp from input table.

Column	Data Type	Description
attribution	DOUBLE PRECISION	Fraction of attribution for conversion event that is attributed to this event.
time_to_conversion	INTEGER	Elapsed time between attributable event and conversion event.

Attribution Example

Event Type Channels

This example uses models to assign attribution weights to these events and channels.

Event Type	Channels
conversion	SocialNetwork, PaidSearch
excluding	Email
optional	Direct, Referral, OrganicSearch

Input

attribution_sample_table1

user_id	event	time_stamp
1	impression	2001-09-27 23:00:01
1	impression	2001-09-27 23:00:05
1	Email	2001-09-27 23:00:15
2	impression	2001-09-27 23:00:31
2	impression	2001-09-27 23:00:51

attribution_sample_table2

user_id	event	time_stamp
1	impression	2001-09-27 23:00:19
1	SocialNetwork	2001-09-27 23:00:20
1	Direct	2001-09-27 23:00:21
1	Referral	2001-09-27 23:00:22
1	PaidSearch	2001-09-27 23:00:23
2	impression	2001-09-27 23:00:29

user_id	event	time_stamp
2	impression	2001-09-27 23:00:31
2	impression	2001-09-27 23:00:33
2	impression	2001-09-27 23:00:36
2	impression	2001-09-27 23:00:38

conversion_event_table

conversion_events
PaidSearch
SocialNetwork

excluding_event_table

excluding_events
Email

optional_event_table

optional_events
Direct
OrganicSearch
Referral

The following two model tables apply the distribution models by rows and by seconds, respectively.

model1_table

id	model
0	SEGMENT_ROWS
1	3:0.5:EXPONENTIAL:0.5,SECOND
2	4:0.3:WEIGHTED:0.4,0.3,0.2,0.1
3	3:0.2:FIRST_CLICK:NA

model2_table

id	model
0	SEGMENT_SECONDS

id	model
1	6:0.5:UNIFORM:NA
2	8:0.3:LAST_CLICK:NA
3	6:0.2:FIRST_CLICK:NA

SQL Call

```
SELECT * FROM Attribution (
  ON attribution_sample_table1 AS InputTable1
    PARTITION BY user_id ORDER BY time_stamp
  ON attribution_sample_table2 AS InputTable2
    PARTITION BY user_id ORDER BY time_stamp
  ON conversion_event_table AS ConversionEventTable DIMENSION
  ON excluding_event_table AS ExcludedEventTable DIMENSION
  ON optional_event_table AS OptionalEventTable DIMENSION
  ON model1_table AS FirstModel DIMENSION
  ON model2_table AS SecondModel DIMENSION
  USING
    EventColumn ('event')
    TimeColumn ('time_stamp')
    WindowSize ('rows:10&seconds:20')
) AS dt ORDER BY user_id, time_stamp;
```

Output

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0.285714	-19
1	impression	2001-09-27 23:00:05	0	
1	impression	2001-09-27 23:00:19	0.714286	-1
1	SocialNetwork	2001-09-27 23:00:20		
1	Direct	2001-09-27 23:00:21	0.5	-2
1	Referral	2001-09-27 23:00:22	0.5	-1
1	PaidSearch	2001-09-27 23:00:23		

DecisionForestPredict

The DecisionForestPredict function uses the model output by ML Engine DecisionForest function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

Note:

DecisionForestPredict outputs the probability that each observation is in the *predicted* class. To use DecisionForestPredict output as input to ML Engine ROC function, you must first transform it to show the probability that each observation is in the *positive* class. One way to do this is to change the probability to (1- current probability) when the predicted class is negative.

Note:

The prediction algorithm compares floating-point numbers. Due to possible inherent data type differences between ML Engine and NewSQL Engine executions, predictions can differ. Before calling the function, compute the relative error, using this formula:

$$\text{relative_error} = (\text{abs}(\text{mle_prediction} - \text{td_prediction}) / \text{mle_prediction}) * 100$$

where *mle_prediction* is ML Engine prediction value and *td_prediction* is NewSQL Engine prediction value. Errors (e) follow Gaussian law; $0 < e < 3\%$ is a negligible difference, with high confidence.

DecisionForestPredict Syntax

```
SELECT * FROM DecisionForestPredict (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('id_column')
  [ NumericInputs ('numeric_input_column'[,...]) ]
  [ CategoricalInputs ('categorical_input_column'[,...]) ]
  [ Detailed ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Accumulate ('accumulate_column'[,...])]
) AS alias;
```

DecisionForestPredict Syntax Elements

IDColumn

Specify the column that contains a unique identifier for each test point in the test set.

NumericInputs

[Optional] Specify the names of the columns that contain the numeric predictor variables.

Default behavior: The function gets these variables from the model output by DecisionForest only if you omit both NumericInputs and CategoricalInputs. If you specify this syntax element, you must specify it exactly as you specified it in the DecisionForest call that created the model.

CategoricalInputs

[Optional] Specify the names of the columns that contain the categorical predictor variables.

Default behavior: The function gets these variables from the model output by DecisionForest only if you omit both NumericInputs and CategoricalInputs. If you specify this syntax element, you must specify it exactly as you specified it in the DecisionForest call that created the model.

Detailed

[Optional] Specify whether to output detailed information about the forest trees; that is, the decision tree and the specific tree information, including task index and tree index for each tree.

Default: 'false'

Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

DecisionForestPredict Input

Table	Description
Input	Contains test data.
Model	[Optional] Has same schema as OutputTable of ML Engine DecisionForest function.

Input Table Schema

Column	Data Type	Description
<i>id_column</i>	Any	Unique test point identifier. Cannot be NULL.
<i>numeric_column</i>	NUMERIC, INTEGER, BIGINT, or DOUBLE PRECISION	[Column appears once for each specified <i>numeric_input_column</i> .] Numeric predictor variable. Cannot be NULL.
<i>category_column</i>	INTEGER, BIGINT, or VARCHAR	[Column appears once for each specified <i>categorical_input_column</i> .] Categorical predictor variable. Cannot be NULL.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
worker_ip	VARCHAR	IP address of worker that produced decision tree.
task_index	INTEGER, BIGINT, or SMALLINT	Identifier of worker that produced decision tree.
tree_num	INTEGER, BIGINT, or SMALLINT	Decision tree identifier.

Column	Data Type	Description
tree	VARCHAR, CLOB, or JSON	JSON representation of decision tree.

DecisionForestPredict Output

Output Table Schema

The table has a set of predictions for each test point.

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	[Column appears once for each specified <i>accumulate_column</i> .]] Column copied from input table.
<i>id_column</i>	Same as in input table	Column copied from input table. Unique row identifier.
<i>prediction</i>	VARCHAR	Predicted test point value, predicted by model.
confidence_lower	DOUBLE PRECISION	Lower bound of confidence interval. For classification tree, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
confidence_upper	DOUBLE PRECISION	Upper bound of confidence interval. For classification tree, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
tree_num	VARCHAR	Either the concatenation of task_index and tree_num from the model table, to show which tree created the prediction, or 'final' to show the overall prediction. This column appears only if you specify Detailed ('true').

DecisionForestPredict Example

Input

- Input table: housing_test, which has 54 observations of 14 variables
- Model: rft_model, output by "DecisionForest Example: TreeType ('classification') and OutOfBag ('false')" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

Input Table Column Descriptions

Column	Description
sn	Sale number (unique identifier of observation)
price	Sale price in U. S. dollars (numeric)

Column	Description
lotsize	Lot size in square feet (numeric)
bedrooms	Number of bedrooms (numeric)
bathrms	Number of full bathrooms (numeric)
stories	Number of stories, excluding basement (numeric)
driveway	Whether the house has a driveway—yes or no (categorical)
recroom	Whether the house has a recreation room—yes or no (categorical)
fullbase	Whether the house has a full finished basement—yes or no (categorical)
gashw	Whether the house uses gas to heat water—yes or no (categorical)
airco	Whether the house has central air conditioning—yes or no (categorical)
garagepl	Number of garage places (numeric)
prefarea	Whether the house is in a preferred neighborhood—yes or no (categorical)
homestyle	Style of house (response variable)

housing_test

sn	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	q
13	27000	1700	3	1	2	yes	no	no	no	no	0
16	37900	3185	2	1	1	yes	no	no	no	yes	0
25	42000	4960	2	1	1	yes	no	no	no	no	0
38	67000	5170	3	1	4	yes	no	no	no	yes	0
53	68000	9166	2	1	1	yes	no	yes	no	yes	2
104	132000	3500	4	2	2	yes	no	no	yes	no	2
111	43000	5076	3	1	1	no	no	no	no	no	0
117	93000	3760	3	1	2	yes	no	no	yes	no	2
132	44500	3850	3	1	2	yes	no	no	no	no	0
140	43000	3750	3	1	2	yes	no	no	no	no	0
142	40000	2650	3	1	2	yes	no	yes	no	no	1
157	60000	2953	3	1	2	yes	no	yes	no	yes	0
...

rtf_model

worker_ip	task_index	tree_num	CAST(tree AS VARCHAR(50))
xx.xx.xx.xx	0	0	{"responseCounts_":{"Eclectic":148,"bungalow":30,"
xx.xx.xx.xx	0	1	{"responseCounts_":{"Eclectic":158,"bungalow":26,"
xx.xx.xx.xx	0	2	{"responseCounts_":{"Eclectic":120,"bungalow":38,"
xx.xx.xx.xx	0	3	{"responseCounts_":{"Eclectic":166,"bungalow":29,"
xx.xx.xx.xx	0	4	{"responseCounts_":{"Eclectic":138,"bungalow":32,"
xx.xx.xx.xx	0	5	{"responseCounts_":{"Eclectic":158,"bungalow":34,"
xx.xx.xx.xx	0	6	{"responseCounts_":{"Eclectic":168,"bungalow":32,"
xx.xx.xx.xx	0	7	{"responseCounts_":{"Eclectic":145,"bungalow":40,"
xx.xx.xx.xx	0	8	{"responseCounts_":{"Eclectic":150,"bungalow":34,"
xx.xx.xx.xx	0	9	{"responseCounts_":{"Eclectic":156,"bungalow":42,"
xx.xx.xx.xx	0	10	{"responseCounts_":{"Eclectic":148,"bungalow":18,"
xx.xx.xx.xx	0	11	{"responseCounts_":{"Eclectic":147,"bungalow":20,"
xx.xx.xx.xx	0	12	{"responseCounts_":{"Eclectic":150,"bungalow":31,"
xx.xx.xx.xx	0	13	{"responseCounts_":{"Eclectic":135,"bungalow":32,"
xx.xx.xx.xx	0	14	{"responseCounts_":{"Eclectic":139,"bungalow":24,"
xx.xx.xx.xx	0	15	{"responseCounts_":{"Eclectic":146,"bungalow":27,"
xx.xx.xx.xx	0	16	{"responseCounts_":{"Eclectic":152,"bungalow":23,"
xx.xx.xx.xx	0	17	{"responseCounts_":{"Eclectic":135,"bungalow":23,"
xx.xx.xx.xx	0	18	{"responseCounts_":{"Eclectic":148,"bungalow":29,"
xx.xx.xx.xx	0	19	{"responseCounts_":{"Eclectic":166,"bungalow":33,"
xx.xx.xx.xx	0	20	{"responseCounts_":{"Eclectic":142,"bungalow":28,"
xx.xx.xx.xx	0	21	{"responseCounts_":{"Eclectic":172,"bungalow":27,"
xx.xx.xx.xx	0	22	{"responseCounts_":{"Eclectic":147,"bungalow":37,"
xx.xx.xx.xx	0	23	{"responseCounts_":{"Eclectic":158,"bungalow":31,"
xx.xx.xx.xx	0	24	{"responseCounts_":{"Eclectic":158,"bungalow":33,"
xx.xx.xx.xx	1	0	{"responseCounts_":{"Eclectic":140,"bungalow":44,"
xx.xx.xx.xx	1	1	{"responseCounts_":{"Eclectic":161,"bungalow":28,"
xx.xx.xx.xx	1	2	{"responseCounts_":{"Eclectic":131,"bungalow":25,"

worker_ip	task_index	tree_num	CAST(tree AS VARCHAR(50))
xx.xx.xx.xx	1	3	{"responseCounts_":{"Eclectic":167,"bungalow":28,"
xx.xx.xx.xx	1	4	{"responseCounts_":{"Eclectic":150,"bungalow":19,"
xx.xx.xx.xx	1	5	{"responseCounts_":{"Eclectic":158,"bungalow":24,"
xx.xx.xx.xx	1	6	{"responseCounts_":{"Eclectic":177,"bungalow":32,"
xx.xx.xx.xx	1	7	{"responseCounts_":{"Eclectic":156,"bungalow":24,"
xx.xx.xx.xx	1	8	{"responseCounts_":{"Eclectic":156,"bungalow":37,"
xx.xx.xx.xx	1	9	{"responseCounts_":{"Eclectic":165,"bungalow":24,"
xx.xx.xx.xx	1	10	{"responseCounts_":{"Eclectic":135,"bungalow":29,"
xx.xx.xx.xx	1	11	{"responseCounts_":{"Eclectic":140,"bungalow":20,"
xx.xx.xx.xx	1	12	{"responseCounts_":{"Eclectic":156,"bungalow":24,"
xx.xx.xx.xx	1	13	{"responseCounts_":{"Eclectic":147,"bungalow":34,"
xx.xx.xx.xx	1	14	{"responseCounts_":{"Eclectic":151,"bungalow":22,"
xx.xx.xx.xx	1	15	{"responseCounts_":{"Eclectic":161,"bungalow":18,"
xx.xx.xx.xx	1	16	{"responseCounts_":{"Eclectic":156,"bungalow":19,"
xx.xx.xx.xx	1	17	{"responseCounts_":{"Eclectic":126,"bungalow":29,"
xx.xx.xx.xx	1	18	{"responseCounts_":{"Eclectic":148,"bungalow":26,"
xx.xx.xx.xx	1	19	{"responseCounts_":{"Eclectic":177,"bungalow":21,"
xx.xx.xx.xx	1	20	{"responseCounts_":{"Eclectic":137,"bungalow":31,"
xx.xx.xx.xx	1	21	{"responseCounts_":{"Eclectic":171,"bungalow":28,"
xx.xx.xx.xx	1	22	{"responseCounts_":{"Eclectic":146,"bungalow":30,"
xx.xx.xx.xx	1	23	{"responseCounts_":{"Eclectic":149,"bungalow":21,"
xx.xx.xx.xx	1	24	{"responseCounts_":{"Eclectic":158,"bungalow":18,"

SQL Call

Use the Accumulate syntax element to pass the homestyle variable, to easily compare the actual and predicted response for each observation.

```
CREATE MULTISET TABLE rf_housing_predict AS (
  SELECT * FROM DecisionForestPredict (
    ON housing_test PARTITION BY ANY
    ON rft_model AS Model DIMENSION
    USING
```

```

NumericInputs ('price ', 'lotsize ', 'bedrooms ', 'bathrms ',
               'stories ', 'garagepl')
CategoricalInputs ('driveway ', 'recroom ', 'fullbase ', 'gashw ',
                  'airco ', 'prefarea')
IdColumn ('sn')
Accumulate ('homestyle')
Detailed ('false')
) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM rf_housing_predict ORDER BY 2;
```

homestyle	sn	prediction	confidence_lower	confidence_upper
Classic	13	Classic	0.6	0.6
Classic	16	Classic	0.56	0.56
Classic	25	Classic	0.54	0.54
Eclectic	38	Eclectic	0.7	0.7
Eclectic	53	Eclectic	0.54	0.54
bungalow	104	bungalow	0.36	0.36
Classic	111	Classic	0.54	0.54
Eclectic	117	Eclectic	0.46	0.46
Classic	132	Classic	0.54	0.54
Classic	140	Classic	0.52	0.52
Classic	142	Eclectic	0.5	0.5
Eclectic	157	Eclectic	0.64	0.64
Eclectic	161	Eclectic	0.74	0.74
bungalow	162	Eclectic	0.46	0.46
Eclectic	176	Eclectic	0.48	0.48
Eclectic	177	Eclectic	0.56	0.56
Classic	195	Classic	0.76	0.76
Classic	198	Classic	0.48	0.48
Eclectic	224	Eclectic	0.56	0.56

homestyle	sn	prediction	confidence_lower	confidence_upper
Classic	234	Classic	0.64	0.64
Classic	237	Classic	0.48	0.48
Classic	239	Classic	0.52	0.52
Classic	249	Classic	0.7	0.7
Classic	251	Classic	0.6	0.6
Eclectic	254	Eclectic	0.66	0.66
Eclectic	255	Eclectic	0.6	0.6
Classic	260	Eclectic	0.5	0.5
Eclectic	274	Eclectic	0.66	0.66
Classic	294	Classic	0.62	0.62
Eclectic	301	Classic	0.56	0.56
Eclectic	306	Eclectic	0.7	0.7
Eclectic	317	Eclectic	0.5	0.5
bungalow	329	Eclectic	0.52	0.52
bungalow	339	bungalow	0.56	0.56
Eclectic	340	Eclectic	0.54	0.54
Eclectic	353	Eclectic	0.44	0.44
Eclectic	355	Classic	0.4	0.4
Eclectic	364	Eclectic	0.54	0.54
bungalow	367	bungalow	0.52	0.52
bungalow	377	Eclectic	0.46	0.46
Eclectic	401	Eclectic	0.56	0.56
Eclectic	403	Eclectic	0.56	0.56
Eclectic	408	Eclectic	0.56	0.56
Eclectic	411	Eclectic	0.54	0.54
Eclectic	440	Eclectic	0.66	0.66
Eclectic	441	Classic	0.5	0.5
Eclectic	443	Classic	0.52	0.52
Classic	459	Classic	0.74	0.74

homestyle	sn	prediction	confidence_lower	confidence_upper
Classic	463	Eclectic	0.56	0.56
Eclectic	469	Eclectic	0.62	0.62
Eclectic	472	Eclectic	0.54	0.54
bungalow	527	Eclectic	0.52	0.52
bungalow	530	Eclectic	0.58	0.58
Eclectic	540	Eclectic	0.42	0.42

Prediction Accuracy

This query returns the prediction accuracy:

```
SELECT (SELECT count(sn) FROM rf_housing_predict
WHERE homestyle = prediction) / (SELECT count(sn)
FROM rf_housing_predict) AS PA;
```

pa
0.7777777777777777778

DecisionTreePredict

The DecisionTreePredict function uses the model output by ML Engine DecisionTree function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

DecisionTreePredict Syntax

```
SELECT * FROM DecisionTreePredict (
  ON { table | view | (query) } AS AttributeTable
  PARTITION BY pid_col [,...] ORDER BY attribute
  ON { table | view | (query) } AS Model DIMENSION
  USING
  AttrTableGroupbyColumns ('gcol'[,...])
  AttrTablePIDColumns ('pid_col'[,...])
  AttrTableValColumn ('value_column')
  OutputResponseProbDist ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})
  Responses ('response'[,...])
```



```
[ Accumulate ('accumulate_column'[,...])
] AS alias;
```

DecisionTreePredict Syntax Elements

AttrTableGroupByColumns

Specify the names of the columns on which AttributeTable is partitioned. Each partition contains one attribute of the input data.

AttrTablePIDColumns

Specify the names of the columns that define the data point identifiers.

AttrTableValColumn

Specify the name of the AttributeTable column that contains the input values.

OutputResponseProbDist

Specify whether to output probabilities. If this value is true, you must specify in the Responses syntax element every label in the AttributeTable table.

Default: 'false'

Responses

[Required if OutputResponseProbDist is true] Specify the labels in the input table.

Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

Note:

If you are using this function to create input for ML Engine ROC function, this syntax element must specify *actual_label*.

DecisionTreePredict Input

Table	Description
AttributeTable	Contains test data. Has same schema as ML Engine DecisionTree InputTable.
Model	Model output by ML Engine DecisionTree function.

AttributeTable Schema

See *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Double quotation marks around some column names are required because the names contain special characters.

Column	Data Type	Description
node_id	INTEGER, SMALLINT, or BIGINT	Node identifier.
node_size	INTEGER, SMALLINT, or BIGINT	Number of objects in node.
"node_gini(p)" or node_gini	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	GINI impurity value for information in node. For ImpurityMeasurement ('gini'), column name is node_gini(p); otherwise, it is node_gini.
"node_entropy(p)" or node_entropy	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Entropy impurity value for the information in the node. For ImpurityMeasurement ('entropy'), column name is node_entropy(p); otherwise, it is node_entropy.
"node_chisq_pv(p)" or node_chisq_pv	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Chi-square impurity value for the information in the node. For ImpurityMeasurement ('chisquare'), column name is node_chisq_pv(p); otherwise, it is node_chisq_pv.
node_label	CHARACTER or VARCHAR	Output category for node.
node_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by node_label.
split_value	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Numeric split value.
"split_gini(p)" or split_gini	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	GINI impurity measurement for information in node after splitting. For ImpurityMeasurement ('gini'), column name is split_gini(p); otherwise, it is split_gini.
"split_entropy(p)" or split_entropy	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Entropy impurity measurement for the information in node after splitting. For ImpurityMeasurement ('entropy'), column name is split_entropy(p); otherwise, it is split_entropy.
"split_chisq_pv(p)" or split_chisq_pv	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Chi-square impurity measurement for information in node after splitting. For ImpurityMeasurement ('chisquare'), column name is split_chisq_pv(p); otherwise, it is split_chisq_pv.
left_id	INTEGER, SMALLINT, or BIGINT	Identifier of left child of node.
left_size	INTEGER, SMALLINT, or BIGINT	Number of objects in left child of node.
left_label	CHARACTER or VARCHAR	Output category for left child of node.
left_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by left_label.

Column	Data Type	Description
right_id	INTEGER, SMALLINT, or BIGINT	Identifier of right child of node.
right_size	INTEGER, SMALLINT, or BIGINT	Number of objects in right child of node.
right_label	CHARACTER or VARCHAR	Output category for right child of node.
right_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by right_label.
left_bucket	CHARACTER or VARCHAR	When split value is categorical attribute, value in left child of node.
right_bucket	CHARACTER or VARCHAR	When split value is categorical attribute, value in right child of node.
left_label_prob_list	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Probability of each label for left child of node.
right_label_prob_list	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Probability of each label for right child of node.
prob_label_order	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Label order probability for left and right children of node.
attribute	CHARACTER or VARCHAR	Split attribute.
node_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by node_label.
left_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by left_label.
right_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by right_label.

DecisionTreePredict Output

Output Table Schema

Column	Data Type	Description
id_column	Any	Data point identifier from attribute_table (DecisionTree InputTable).
pred_label	VARCHAR	Predicted response value for data point.

Column	Data Type	Description
<code>prob_for_label_n</code>	DOUBLE PRECISION	[Column appears once for each possible label.] Probability that data point has <code>label_n</code> .
<code>actual_label</code>	VARCHAR	Copied from <code>attribute_table</code> (DecisionTree InputTable).

DecisionTreePredict Examples

DecisionTreePredict Examples Input

- AttributeTable: `iris_attribute_test`
- Model: `iris_attribute_output`

Both tables are created in "DecisionTree Example: Create Model" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

For input table column descriptions, see [NaiveBayesPredict Example](#).

`iris_attribute_test`

pid	attribute	attrvalue
5	petal_length	1.4
5	petal_width	0.2
5	sepal_length	5
5	sepal_width	3.6
10	petal_length	1.5
10	petal_width	0.1
10	sepal_length	4.9
10	sepal_width	3.1
15	petal_length	1.2
15	petal_width	0.2
15	sepal_length	5.8
15	sepal_width	4
...

iris_attribute_output

node_id	node_size	node_gini(p)	node_entropy	node_chisq_pv	node_label	node_majorvotes	split_value
0	120	0.6666666666666667	1.58496250072116	1	1	40	3
2	80	0.5	1	1	2	40	1.70000004768372
5	39	0.0499671268902038	0.172036949353113	1	2	38	4.90000009536743
6	41	0.0928019036287924	0.281193796432043	1	3	39	4.90000009536743
14	37	0.0525931336742148	0.179256066928321	1	3	36	2.90000009536743
30	24	0.0798611111111112	0.249882292833186	1	3	23	3.20000004768372
61	14	0.13265306122449	0.371232326640875	1	3	13	6.30000019073486

DecisionTreePredict Example: Apply Model to Test Data

Input

See [DecisionTreePredict Examples Input](#).

SQL Call

```
CREATE MULTISET TABLE decisiontree_predict AS (
  SELECT * FROM DecisionTreePredict (
    ON iris_attribute_test AS AttributeTable PARTITION BY pid
    ORDER BY attribute
    ON iris_attribute_output as Model DIMENSION
    USING
    AttrTableGroupbyColumns ('attribute')
    AttrTablePIDColumns ('pid')
    AttrTableValColumn ('attrvalue')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM decisiontree_predict ORDER BY 1;
```

The predict labels 1, 2, and 3 correspond to species setosa, versicolor, and virginica.

pid	pred_label
5	1
10	1
15	1
20	1
25	1
30	1
35	1
40	1
45	1
50	1
55	2
60	2
65	2
70	2
75	2
80	2
85	2
90	2
95	2
100	2
105	3
110	3
115	3
120	2
125	3
130	2
135	2

pid	pred_label
140	3
145	3
150	3

DecisionTreePredict Example: OutputProb, Responses

Input

See [DecisionTreePredict Examples Input](#).

SQL Call

```
SELECT * FROM DecisionTreePredict (
  ON iris_attribute_test AS AttributeTable PARTITION BY pid ORDER BY attribute
  ON iris_attribute_output AS Model DIMENSION
  USING
  AttrTableGroupByColumns ('attribute')
  AttrTablePIDColumns ('pid')
  AttrTableValColumn ('attrvalue')
  OutputResponseProbDist ('true')
  Responses ('1','2','3')
) AS dt ORDER BY pid;
```

Output

pid	pred_label	prob_for_label_1	prob_for_label_2	prob_for_label_3
5	1	0.95348	0.02326	0.02326
10	1	0.95348	0.02326	0.02326
15	1	0.95348	0.02326	0.02326
20	1	0.95348	0.02326	0.02326
25	1	0.95348	0.02326	0.02326
30	1	0.95348	0.02326	0.02326
35	1	0.95348	0.02326	0.02326
40	1	0.95348	0.02326	0.02326
45	1	0.95348	0.02326	0.02326
50	1	0.95348	0.02326	0.02326

pid	pred_label	prob_for_label_1	prob_for_label_2	prob_for_label_3
55	2	0.02632	0.94736	0.02632
60	2	0.02632	0.94736	0.02632
65	2	0.02632	0.94736	0.02632
70	2	0.02632	0.94736	0.02632
75	2	0.02632	0.94736	0.02632
80	2	0.02632	0.94736	0.02632
85	2	0.02632	0.94736	0.02632
90	2	0.02632	0.94736	0.02632
95	2	0.02632	0.94736	0.02632
100	2	0.02632	0.94736	0.02632
105	3	0.06250	0.12500	0.81250
110	3	0.07692	0.07692	0.84616
115	3	0.06250	0.06250	0.87500
120	3	0.14286	0.57143	0.28571
125	3	0.07692	0.07692	0.84616
130	3	0.14286	0.57143	0.28571
135	3	0.14286	0.57143	0.28571
140	3	0.06250	0.12500	0.81250
145	3	0.07692	0.07692	0.84616
150	3	0.25000	0.25000	0.50000

GLMPredict

The GLMPredict function uses the model output by ML Engine GLM function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

GLMPredict Syntax

```
SELECT * FROM GLMPredict (
  ON { table | ( query ) } [ PARTITION BY ANY ]
```



```

ON { table | view | (query) } AS Model DIMENSION
  [ ORDER BY attribute, predictor, category, estimate ]
[ USING
  [ Accumulate ('accumulate_column'[,...])]
  [ Family ('family') ]
  [ LinkFunction ('link') ]
]
) AS alias;

```

Note:

You must specify the keyword USING to use any function syntax element.

GLMPredict Syntax Elements

Accumulate

[Optional] Specify the names of input table columns to copy to the output table.

Family

[Optional] Specify the distribution exponential family.

Note:

If you specify this syntax element, you must give it the same value that you used for the Family syntax element of ML Engine GLM function when you created the model table.

Default: Read from the model table

LinkFunction

[Optional] Specify the link function. For the canonical link functions (default link functions) and the link functions allowed for each exponential family, see the GLM function description in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Note:

If you specify this syntax element, you must give it the same value that you used for the LinkFunction syntax element of ML Engine GLM function when you created the model table.

Default: 'CANONICAL'

GLMPredict Input

Table	Description
Input	Contains new data.
Model	Model output by ML Engine GLM function. For schema, see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003.

Table	Description
	Note: If the GLM call that created the model table specified the Step syntax element, include the optional ORDER BY clause in the GLMPredict call; otherwise, the GLMPredict result is nondeterministic.

Input Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.
<i>dependent_variable_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, DOUBLE PRECISION, VARCHAR(<i>n</i>), CHAR(<i>n</i>)	Dependent/response variables. Cannot be NULL.
<i>predictor_variable_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, DOUBLE PRECISION	[Column appears one or more times.] Independent/predictor variable. Cannot be NULL.

Note:

Any numeric *dependent_variable_column* or *predictor_variable_column* that is expected to be categorical must be cast to VARCHAR.

Model Table Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
attribute	INTEGER	Numeric index of predictor.
predictor	CHARACTER or VARCHAR	Predictor name.
category	CHARACTER or VARCHAR	For categorical predictor, its level. For numeric predictor, NULL.
estimate	DOUBLE PRECISION	Estimated coefficient.
std_error	DOUBLE PRECISION	Standard error of coefficient.
t_score	DOUBLE PRECISION	[Column appears only with Family ('GAUSSIAN').] The t_score follows a t(N-p-1) distribution.
z_score	DOUBLE PRECISION	[Column appears only without Family ('GAUSSIAN').] The z-score follows the N(0,1) distribution.

Column	Data Type	Description
p_value	DOUBLE PRECISION	p-value for z_score. (p-value represents significance of each coefficient.)
significance	CHARACTER or VARCHAR	Significance code for p_value.
family	CHARACTER or VARCHAR	Distribution exponential family, specified by Family syntax element.

GLMPredict Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in <i>input_table</i>	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.
fitted_value	DOUBLE PRECISION	Score of the input data, given by equation $g^{-1}(X\beta)$, where g^{-1} is the inverse link function, X the predictors, and β is the vector of coefficients estimated by the GLM function. For other values of Family, the scores are the expected values of dependent/response variable, conditional on the predictors.

GLMPredict Examples

GLMPredict Example: Logistic Distribution Prediction

This example predicts the admission status of students.

Input

- Input table: `admissions_test`, which has admissions information for 20 students
- Model: `glm_admissions_model`, output by "GLM Example: Logistic Regression Analysis with Intercept" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

Input Table Column Descriptions

Column	Description
id	Student identifier (unique)
masters	Whether student has a masters degree—yes or no (categorical)
gpa	Grade point average on a 4.0 scale (numerical)
stats	Statistical skills—Novice, Beginner, or Advanced (categorical)

Column	Description
programming	Programming skills—Novice, Beginner, or Advanced (categorical)
admitted	Whether student was admitted—1 (yes) or 0 (no)

admissions_test

id	masters	gpa	stats	programming	admitted
50	yes	3.95	Beginner	Beginner	0
51	yes	3.76	Beginner	Beginner	0
52	no	3.7	Novice	Beginner	1
53	yes	3.5	Beginner	Novice	1
54	yes	3.5	Beginner	Advanced	1
55	no	3.6	Beginner	Advanced	1
56	no	3.82	Advanced	Advanced	1
57	no	3.71	Advanced	Advanced	1
58	no	3.13	Advanced	Advanced	1
59	no	3.65	Novice	Novice	1
60	no	4	Advanced	Novice	1
61	yes	4	Advanced	Advanced	1
62	no	3.7	Advanced	Advanced	1
63	no	3.83	Advanced	Advanced	1
64	yes	3.81	Advanced	Advanced	1
65	yes	3.9	Advanced	Advanced	1
66	no	3.87	Novice	Beginner	1
67	yes	3.46	Novice	Beginner	0
68	no	1.87	Advanced	Novice	1
69	no	3.96	Advanced	Advanced	1

SQL Call

```
CREATE MULTISET TABLE glmpredict_admissions AS (
  SELECT * FROM GLMPredict (
    ON admissions_test PARTITION BY ANY
```

```

ON glm_admissions_model AS Model DIMENSION
  ORDER BY attribute, category, predictor, estimate
USING
  Accumulate ('id','masters','gpa','stats','programming','admitted')
  Family ('LOGISTIC')
  LinkFunction ('LOGIT')
) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_admissions ORDER BY 1;
```

Note:

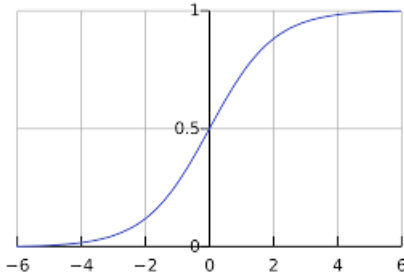
Fitted values can vary in precision, because they depend on the model table output by ML Engine GLM function and fetched to Teradata database.

glmpredict_admissions

id	masters	gpa	stats	programming	admitted	fitted value
50	yes	3.950000000000000E 000	Beginner	Beginner	0	3.50763408888030E-001
51	yes	3.760000000000000E 000	Beginner	Beginner	0	3.55708978581653E-001
52	no	3.700000000000000E 000	Novice	Beginner	1	7.58306140231079E-001
53	yes	3.500000000000000E 000	Beginner	Novice	1	5.56012779663342E-001
54	yes	3.500000000000000E 000	Beginner	Advanced	1	7.69474352959112E-001
55	no	3.600000000000000E 000	Beginner	Advanced	1	9.68031141480050E-001
56	no	3.820000000000000E 000	Advanced	Advanced	1	9.45772725968165E-001
57	no	3.710000000000000E 000	Advanced	Advanced	1	9.46411914806798E-001
58	no	3.130000000000000E 000	Advanced	Advanced	1	9.49666186386367E-001
59	no	3.650000000000000E 000	Novice	Novice	1	8.74189685344822E-001
60	no	4.000000000000000E 000	Advanced	Novice	1	8.65058992199339E-001
61	yes	4.000000000000000E 000	Advanced	Advanced	1	6.50618727191735E-001
62	no	3.700000000000000E 000	Advanced	Advanced	1	9.46469669158547E-001
63	no	3.830000000000000E 000	Advanced	Advanced	1	9.45714262806374E-001
64	yes	3.810000000000000E 000	Advanced	Advanced	1	6.55523357798207E-001
65	yes	3.900000000000000E 000	Advanced	Advanced	1	6.53204164468356E-001
66	no	3.870000000000000E 000	Novice	Beginner	1	7.54738501228429E-001
67	yes	3.460000000000000E 000	Novice	Beginner	0	2.60034297764359E-001
68	no	1.870000000000000E 000	Advanced	Novice	1	8.90965518431337E-001
69	no	3.960000000000000E 000	Advanced	Advanced	1	9.44948816395031E-001

Categorizing fitted_value Column

The fitted_value column gives the probability that a student belongs to one of the output classes. The following figure shows a typical logistic regression graph, mapping the input x-axis against a y probability value between [0,1].



A fitted_value probability greater than or equal to 0.5 implies class 1 (student admitted); a probability less than 0.5 implies class 0 (student rejected).

The following code adds a fitted_category column to glmpredict_admissions and populates it:

```
ALTER table glmpredict_admissions
  ADD fitted_category int;
UPDATE glmpredict_admissions SET fitted_category = 1
  WHERE fitted_value > 0.4999;
UPDATE glmpredict_admissions SET fitted_category = 0
  WHERE fitted_value < 0.4999;
```

This query returns the following table:

```
SELECT * FROM glmpredict_admissions ORDER BY 1;
```

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
50	yes	3.950000000000000E000	Beginner	Beginner	0	3.50763408888030E-001	0
51	yes	3.760000000000000E000	Beginner	Beginner	0	3.55708978581653E-001	0
52	no	3.700000000000000E000	Novice	Beginner	1	7.58306140231079E-001	1
53	yes	3.500000000000000E000	Beginner	Novice	1	5.56012779663342E-001	1

2: Teradata Vantage NewSQL Engine Analytic Functions

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
54	yes	3. 50000000000000E 000	Beginner	Advanced	1	7. 69474352959112E-001	1
55	no	3. 60000000000000E 000	Beginner	Advanced	1	9. 68031141480050E-001	1
56	no	3. 82000000000000E 000	Advanced	Advanced	1	9. 45772725968165E-001	1
57	no	3. 71000000000000E 000	Advanced	Advanced	1	9. 46411914806798E-001	1
58	no	3. 13000000000000E 000	Advanced	Advanced	1	9. 49666186386367E-001	1
59	no	3. 65000000000000E 000	Novice	Novice	1	8. 74189685344822E-001	1
60	no	4. 00000000000000E 000	Advanced	Novice	1	8. 65058992199339E-001	1
61	yes	4. 00000000000000E 000	Advanced	Advanced	1	6. 50618727191735E-001	1
62	no	3. 70000000000000E 000	Advanced	Advanced	1	9. 46469669158547E-001	1
63	no	3. 83000000000000E 000	Advanced	Advanced	1	9. 45714262806374E-001	1
64	yes	3. 81000000000000E 000	Advanced	Advanced	1	6. 55523357798207E-001	1
65	yes	3. 90000000000000E 000	Advanced	Advanced	1	6. 53204164468356E-001	1
66	no	3. 87000000000000E 000	Novice	Beginner	1	7. 54738501228429E-001	1
67	yes	3. 46000000000000E 000	Novice	Beginner	0	2. 60034297764359E-001	0

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
68	no	1. 870000000000000E 000	Advanced	Novice	1	8. 90965518431337E-001	1
69	no	3. 960000000000000E 000	Advanced	Advanced	1	9. 44948816395031E-001	1

Prediction Accuracy

This query returns the prediction accuracy:

```
SELECT (SELECT COUNT(id) FROM glmpredict_admissions
WHERE admitted = fitted_category)/(SELECT count(id)
FROM glmpredict_admissions) AS prediction_accuracy;
```

prediction_accuracy
1.00000000000000000000

GLMPredict Example: Gaussian Distribution Prediction

This example evaluates the predictions for new houses, comparing the original price information with root mean square error evaluation (RMSE).

Input

- Input table: housing_test, as in [DecisionForestPredict Example](#)
- Model: glm_housing_model, output by "GLM Example: Gaussian Distribution Analysis" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

SQL Call

The canonical link specifies the default family link, which is "identity" for the Gaussian distribution.

```
DROP TABLE glmpredict_housing;

CREATE MULTISET TABLE glmpredict_housing AS (
  SELECT * FROM GLMPredict (
    ON housing_test PARTITION BY ANY
    ON glm_housing_model AS Model DIMENSION
    ORDER BY attribute, category, predictor, estimate
    USING
    Accumulate ('sn', 'price')
```



```

    Family ('GAUSSIAN')
    LinkFunction ('CANONICAL')
  ) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_housing ORDER BY 1;
```

sn	price	fitted_value
13	27000	3.73458440000000E 004
16	37900	4.36871317500000E 004
25	42000	4.09020280000000E 004
38	67000	7.24876705000000E 004
53	68000	7.92386937000000E 004
104	132000	1.11528007000000E 005
111	43000	3.91028812000000E 004
117	93000	6.69369510000000E 004
132	44500	4.18198865000000E 004
140	43000	4.16117915000000E 004
142	40000	4.43941465000000E 004
157	60000	6.65712643500000E 004
161	63900	6.49009829000000E 004

The fitted_value column gives the predicted house price.

Root Mean Square Error Evaluation

This query returns the root mean square error evaluation (RMSE):

```
SELECT SQRT(AVG(POWER(glmpredict_housing.price -
glmpredict_housing.fitted_value, 2))) AS RMSE FROM glmpredict_housing;
```

rmse
1.06854695738768E 004

GLMPredict Example: Casting Input Column to VARCHAR

Like [GLMPredict Example: Logistic Distribution Prediction](#), this example predicts the admission status of students. In both examples, the input column masters is categorical—the value can be yes or no. In the other example, the value is 'yes' or 'no'. In this example, the value is numerical—1 for yes or 0 for no—therefore, it must be cast to VARCHAR.

Input

- Input table: admissions_test_2, which has admissions information for 20 students
- Model: glm_admissions_model, output by "GLM Example: Logistic Regression Analysis with Intercept" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003, with the category column modified as follows:

attribute	predictor	category
1	masters	'1'
2	masters	'0'

admissions_test_2

id	masters	gpa	stats	programming	admitted
50	1	3.950000000000000E 000	Beginner	Beginner	0
51	1	3.760000000000000E 000	Beginner	Beginner	0
52	0	3.700000000000000E 000	Novice	Beginner	1
53	1	3.500000000000000E 000	Beginner	Novice	1
54	1	3.500000000000000E 000	Beginner	Advanced	1
55	0	3.600000000000000E 000	Beginner	Advanced	1
56	0	3.820000000000000E 000	Advanced	Advanced	1
57	0	3.710000000000000E 000	Advanced	Advanced	1
58	0	3.130000000000000E 000	Advanced	Advanced	1
59	0	3.650000000000000E 000	Novice	Novice	1
60	0	4.000000000000000E 000	Advanced	Novice	1
61	1	4.000000000000000E 000	Advanced	Advanced	1
62	0	3.700000000000000E 000	Advanced	Advanced	1
63	0	3.830000000000000E 000	Advanced	Advanced	1
64	1	3.810000000000000E 000	Advanced	Advanced	1

id	masters	gpa	stats	programming	admitted
65	1	3.900000000000000E 000	Advanced	Advanced	1
66	0	3.870000000000000E 000	Novice	Beginner	1
67	1	3.460000000000000E 000	Novice	Beginner	0
68	0	1.870000000000000E 000	Advanced	Novice	1
69	0	3.960000000000000E 000	Advanced	Advanced	1

SQL Call

```
CREATE MULTISET TABLE glmpredict_admissions_2 AS (
  SELECT * FROM GLMPredict (
    ON (
      SELECT id, CAST(masters AS varchar(10)) AS masters,
        gpa, stats, programming, admitted
      FROM admissions_test
    ) PARTITION BY ANY
    ON glm_admissions_model AS Model DIMENSION
    ORDER BY attribute, category, predictor, estimate
    USING
    Accumulate ('id','masters','gpa','stats','programming','admitted')
    Family ('LOGISTIC')
    LinkFunction ('LOGIT')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_admissions_2 ORDER BY 1;
```

Note:

Fitted values can vary in precision, because they depend on the model table output by ML Engine GLM function and fetched to Teradata database.

glmpredict_admissions_2

id	masters	gpa	stats	programming	admitted	fitted value
50	1	3.950000000000000E 000	Beginner	Beginner	0	3.50763408888030E-001
51	1	3.760000000000000E 000	Beginner	Beginner	0	3.55708978581653E-001
52	0	3.700000000000000E 000	Novice	Beginner	1	7.58306140231079E-001
53	1	3.500000000000000E 000	Beginner	Novice	1	5.56012779663342E-001

id	masters	gpa	stats	programming	admitted	fitted value
54	1	3.500000000000000E 000	Beginner	Advanced	1	7.69474352959112E-001
55	0	3.600000000000000E 000	Beginner	Advanced	1	9.68031141480050E-001
56	0	3.820000000000000E 000	Advanced	Advanced	1	9.45772725968165E-001
57	0	3.710000000000000E 000	Advanced	Advanced	1	9.46411914806798E-001
58	0	3.130000000000000E 000	Advanced	Advanced	1	9.49666186386367E-001
59	0	3.650000000000000E 000	Novice	Novice	1	8.74189685344822E-001
60	0	4.000000000000000E 000	Advanced	Novice	1	8.65058992199339E-001
61	1	4.000000000000000E 000	Advanced	Advanced	1	6.50618727191735E-001
62	0	3.700000000000000E 000	Advanced	Advanced	1	9.46469669158547E-001
63	0	3.830000000000000E 000	Advanced	Advanced	1	9.45714262806374E-001
64	1	3.810000000000000E 000	Advanced	Advanced	1	6.55523357798207E-001
65	1	3.900000000000000E 000	Advanced	Advanced	1	6.53204164468356E-001
66	0	3.870000000000000E 000	Novice	Beginner	1	7.54738501228429E-001
67	1	3.460000000000000E 000	Novice	Beginner	0	2.60034297764359E-001
68	0	1.870000000000000E 000	Advanced	Novice	1	8.90965518431337E-001
69	0	3.960000000000000E 000	Advanced	Advanced	1	9.44948816395031E-001

Categorizing fitted_value Column

See [GLMPredict Example: Logistic Distribution Prediction](#).

Prediction Accuracy

See [GLMPredict Example: Logistic Distribution Prediction](#).

MovingAverage

The MovingAverage function computes average values in a series, using the specified moving average type.

Moving Average Type	Description
Cumulative Moving Average	Computes cumulative moving average of value from beginning of series.
Exponential Moving Average	Computes average of points in series, applying damping factor that exponentially decreases weights of older values.
Modified Moving Average	Computes first value as simple moving average. Computes subsequent values by adding new value and subtracting last average from resulting sum.
Simple Moving Average	Computes unweighted mean of previous n data points.
Triangular Moving Average	Computes double-smoothed average of points in series.

Moving Average Type	Description
Weighted Moving Average	Computes average of points in series, applying weights to older values. Weights for older values decrease arithmetically.

Note:

- This function requires the UTF8 client character set.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

- The ORDER BY clause supports only ASCII collation.
- The PARTITION BY clause assumes column names are in Normalization Form C (NFC).

Weighted Moving Average

A weighted average has multiplying factors that give different weights to different data points. Mathematically, the moving average is the convolution of the data points with a moving average function. In technical analysis, a weighted moving average (WMA) has weights that decrease arithmetically. In an n -point weighted moving average, the most recent data point has weight n , the second most recent data point has weight $(n - 1)$, and so on, until the weight is zero.

With MAvgType ('W'), the MovingAverage function uses this formula:

$$WMA_M = (nV_M + (n-1)V_{M-1} + \dots + 2V_{(M-n+2)} + V_{(M-n+1)}) / (n + (n-1) + \dots + 2 + 1)$$

Where $Total_M = V_M + \dots + V_{(M-n+1)}$, the following equations are true:

- $Total_{M+1} = Total_M + V_{M+1} - V_{(M-n+1)}$
- $Numerator_{M+1} = Numerator_M + n*V_{M+1} - Total_M$
- $WMA_{M+1} = Numerator_{M+1} / (n*(n+1)/2)$

V_M is the target column value at index M in the window under consideration.

The value n —the number of old values to use when calculating the new weighted moving average—is specified by the WindowSize syntax element.

Triangular Moving Average

The triangular moving average (TMA) differs from the simple moving average in that it is double-smoothed; that is, averaged twice. Double-smoothing keeps the triangular moving average from responding to new data points as fast as the simple moving average. For an average that responds quickly to new data points, use the simple moving average or exponential moving average.

With MAvgType ('T'), the MovingAverage function uses this procedure:

1. Compute the window size, N :

$$N = \text{ceil}(\text{window_size} + 1) / 2$$

The value *window_size* is specified by the WindowSize syntax element.

2. Compute the simple moving average of each target column, using this formula:

$$\text{SMA}_i = (V_1 + V_2 + \dots + V_N) / N$$

The function calculates SMA_i on the i th window of the target column from the start of the row.

V_i is the value of the target column at index i in the window.

3. Compute the triangular moving average by computing the simple moving average with window size N on the values obtained in step 2, using this formula:

$$\text{TMA} = (\text{SMA}_1 + \text{SMA}_2 + \dots + \text{SMA}_N) / N$$

Note:

The function writes the cumulative moving average values computed for the first n rows, where n is less than N , to the output table.

Simple Moving Average

The simple moving average (SMA) is the unweighted mean of the previous n data points. For example, a 10-day simple moving average of closing price is the mean of closing prices for the previous 10 days.

With MAvgType ('S'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *window_size* rows.

The value *window_size* is specified by the WindowSize syntax element. In the next step, it is called N .

2. For each subsequent row, compute the new simple moving average value with this formula:

$$\text{SMA} = (V_1 + V_2 + \dots + V_N) / N$$

V_i is the value of the target column at index i in the window.

Modified Moving Average

The first point of the modified moving average (MMA) is calculated like the first point of the simple moving average. Each subsequent point is calculated by adding the new value to the most recently calculated modified moving average value and then, from that sum, subtracting the last average value. The difference is the new modified moving average value.

With MAvgType ('M'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *window_size* rows.

The value *window_size* is specified by the WindowSize syntax element.

2. For each subsequent row, compute the new modified moving average value with this formula:

$$MMA_M = MMA_{M-1} + (1/window_size) (V_M - MMA_{M-1})$$

V_M is the current value.

Exponential Moving Average

Exponential moving average (EMA), or exponentially weighted moving average (EWMA), applies a damping factor, *alpha*, that exponentially decreases the weights of older values. This technique gives much more weight to recent observations, while retaining older observations.

With MAvgType ('E'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first n rows.

The value n is specified by the StartRows syntax element.

2. For each subsequent row, compute the new exponential moving average value with this formula:

$$EMA_M = \alpha * V + (1 - \alpha) * EMA_{M-1}$$

The value *alpha* is specified by the Alpha syntax element. V is the new value.

Cumulative Moving Average

In a cumulative moving average (CMA), the data are added to the data set in an ordered data stream over time. The objective is to compute the average of all the data at each point in time when new data arrived. A typical use case is an investor who wants to find the average price of all transactions of a specific stock over time, up to the current time.

With MAvgType ('C'), the MovingAverage function computes the arithmetic average of all the rows from the beginning of the series with this formula:

$$CMA = (V_1 + V_2 + \dots + V_N)/N$$

V_i is a value. N is the number of rows from the beginning of the data set.

MovingAverage Syntax

```
SELECT * FROM MovingAverage (
  ON { table | view | (query) }
    [ PARTITION BY partition_column [,...] ]
    [ ORDER BY order_column [,...] ]
  [ USING
    [ MAvgType ({ 'C' | 'E' | 'M' | 'S' | 'T' | 'W' }) ]
    [ TargetColumns ('target_column' [,...]) ]
    [ Alpha (alpha) ]
    [ StartRows (n) ]
    [ IncludeFirst ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
    [ WindowSize (window_size) ]
```

```
]
) AS alias;
```

Note:

If the ON clause does not include the PARTITION BY and ORDER BY clauses, results are nondeterministic.

MovingAverage Syntax Elements

MAvgType

[Optional] Specify one of the following moving average types:

Type	Description
'C' (Default)	Cumulative moving average.
'E'	Exponential moving average.
'M'	Modified moving average.
'S'	Simple moving average.
'T'	Triangular moving average.
'W'	Weighted moving average.

TargetColumns

[Optional] Specify the input column names for which to compute the moving average.

Default behavior: The function copies every input column to the output table but does not compute any moving averages.

Alpha

[Optional with MAvgType E, otherwise ignored.] Specify the damping factor, a value in the range [0, 1], which represents a percentage in the range [0, 100]. For example, if *alpha* is 0.2, the damping factor is 20%. A higher *alpha* discounts older observations faster.

Default: 0.1

StartRows

[Optional with MAvgType E, otherwise ignored.] Specify the number of rows to skip before calculating the exponential moving average. The function uses the arithmetic average of these rows as the initial value of the exponential moving average. The value *n* must be an integer.

Default: 2

IncludeFirst

[Ignored with MAvgType C, otherwise optional.] Specify whether to include the starting rows in the output table. If you specify 'true', the output columns for the starting rows contain NULL, because their moving average is undefined.

Default: 'false'

WindowSize

[Optional with MAvgType M, S, T, and W; otherwise ignored.] Specify the number of previous values to consider when computing the new moving average. The data type of *window_size* must be BYTEINT, SMALLINT, or INTEGER.

Minimum value: 3

Default: '10'

MovingAverage Input

Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned. This column must contain all rows of an entity. For example, if function is to compute moving average of a particular stock share price, all transactions of that stock must be in one partition. Note: PARTITION BY clause assumes column names are in Normalization Form C (NFC).
<i>order_column</i>	Any	[Column appears once for each specified <i>order_column</i> .] Column by which input table is ordered. Note: ORDER BY clause supports only ASCII collation.
<i>target_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMBER, or DOUBLE PRECISION	[Column appears once for each specified <i>target_column</i> .] Values to average.

MovingAverage Output

Output Table Schema

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	Column by which input data is partitioned.
<i>order_column</i>	Same as in input table	Column by which input table is ordered.
<i>target_column</i>	Same as in input table	Column copied from input table.

Column	Data Type	Description
<i>target_column_</i> <i>typemavg</i>	DOUBLE PRECISION	Moving average of <i>target_column</i> values when row was added to data set, where <i>type</i> is value of MAvgType syntax element.

MovingAverage Examples

MovingAverage Examples Input

The input table, `company1_stock`, contains 10 observations of common stock closing prices.

company1_stock

id	name	period	stockprice
1	Company1	1961-05-17 00:00:00.000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000
10	Company1	1961-05-31 00:00:00.000000	490.000000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000

id	name	period	stockprice
20	Company1	1961-06-14 00:00:00.000000	491.000000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000
22	Company1	1961-06-16 00:00:00.000000	482.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000

MovingAverage Example: Cumulative Moving Average

This example computes a cumulative moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('C')
  TargetColumns ('stockprice')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_cmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	458.500000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	456.333333333333
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	458.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.166666666667
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	458.857142857143
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.375000000000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.888888888889

id	name	period	stockprice	stockprice_cmavg
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.636363636364
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	474.076923076923
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	475.714285714286
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	477.066666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	477.875000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	478.529411764706
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	478.500000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	478.947368421053
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	479.550000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	479.904761904762
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	480.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	479.956521739130
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	479.875000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	479.840000000000

MovingAverage Example: Exponential Moving Average

This example computes an exponential moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('E')
  TargetColumns ('stockprice')
  StartRows (10)
  Alpha (0.1)
```

```
IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_emavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.860000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.674000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.306600000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.475940000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.328346000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.395511400000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.255960260000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.930364234000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.537327810600
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.483595029540
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.935235526586
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.841711973927
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.457540776535
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	482.011786698881
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.710608028993

MovingAverage Example: Modified Moving Average

This example computes the triangular moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('M')
  TargetColumns ('stockprice')
  WindowSize (10)
  IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_mmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.700000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.930000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	458.937000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	459.243300000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	459.218970000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	459.597073000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.537365700000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.683629130000
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.215266217000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.693739595300
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.524365635770
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.171929072193
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.354736164974
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.219262548476

id	name	period	stockprice	stockprice_mmavg
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.297336293629
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.167602664266
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.850842397839
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.465758158055
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.419182342250
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.877264108025
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.789537697222
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.410583927500
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	481.969525534750
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.672572981275

MovingAverage Example: Simple Moving Average

This example computes a simple moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('S')
  TargetColumns ('stockprice')
  WindowSize (10)
  IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_smavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	

id	name	period	stockprice	stockprice_smaavg
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	470.600000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	474.700000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	479.400000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	483.200000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.600000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	489.700000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	492.300000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	492.200000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	491.600000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	491.700000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.200000000000
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.600000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.600000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	485.700000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	484.000000000000

MovingAverage Example: Triangular Moving Average

This example computes the triangular moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```

SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('T')
  TargetColumns ('stockprice')
  WindowSize (10)
  IncludeFirst ('true')
) AS dt ORDER BY id;

```

Output

id	name	period	stockprice	stockprice_tmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.250000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.277777777778
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.958333333333
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	457.966666666667
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	457.777777777778
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	458.416666666667
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	460.555555555556
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	463.444444444444
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	467.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	471.611111111111
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	477.138888888889
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	482.555555555556
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.916666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	490.416666666667
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	493.944444444444
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	493.555555555556
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	492.500000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.111111111111

id	name	period	stockprice	stockprice_tmavg
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.500000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.694444444444
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	486.444444444444
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	485.305555555556

MovingAverage Example: Weighted Moving Average

This example computes the weighted moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
    MAvgType ('W')
    TargetColumns ('stockprice')
    WindowSize (10)
    IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_wmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	

id	name	period	stockprice	stockprice_wmavg
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	473.454545454545
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	477.927272727273
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	482.909090909091
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	487.327272727273
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	490.527272727273
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	492.854545454545
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	493.472727272727
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.345454545455
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	490.745454545455
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	489.800000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	489.690909090909
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	488.836363636364
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	487.163636363636
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	485.236363636364
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	483.490909090909
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	482.272727272727

NaiveBayesPredict

The NaiveBayesPredict function uses the model output by ML Engine Naive Bayes Classifier function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

NaiveBayesPredict Syntax

```
SELECT * FROM NaiveBayesPredict (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('test_point_id_col')
  NumericInputs ('numeric_input_column'[,...])
  CategoricalInputs ('categorical_input_column'[,...]) )
```

```
Responses ('response'[,...])
) AS alias;
```

NaiveBayesPredict Syntax Elements

IDColumn

Specify the name of the input table column that contains the ID that uniquely identifies the test input data.

NumericInputs

[Required if CategoricalInputs is omitted.] Specify the same *numeric_input_columns* that you specified when you used the NaiveBayesMap and NaiveBayesReduce functions to create the model table from the training data.

CategoricalInputs

[Required if NumericInputs is omitted.] Specify the same *categorical_input_columns* that you specified when you used the NaiveBayesMap and NaiveBayesReduce functions to create the model table from the training data.

Responses

Specify the responses to output.

NaiveBayesPredict Input

Table	Description
Input	Contains test data. Has same schema as ML Engine Naive Bayes Classifier input table.
Model	Model output by ML Engine Naive Bayes Classifier function.

Input Table Schema

See *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Double quotation marks around some column names are required because the names are either Teradata Database reserved keywords or are camel-case.

Column	Data Type	Description
"class" or class_nb	VARCHAR	Response.
"variable" or variable_nb	VARCHAR	Input variable (name of input column).
"type" or type_nb	VARCHAR	Input variable types ('NUMERIC' or 'CATEGORICAL').

Column	Data Type	Description
category	VARCHAR	For categorical predictor, its level. For numeric predictor, NULL.
cnt	INTEGER or BIGINT	Count of observations with this class, variable, and category.
"sum" or sum_nb	INTEGER or DOUBLE PRECISION	For numerical predictor, sum of variable values for observations with this class, variable, and category. For categorical predictor, NULL.
"sumSq" or sum_sq	INTEGER or DOUBLE PRECISION	For numerical predictor, sum of square of variable values for observations with this class, variable, and category. For categorical predictor, NULL.
"totalCnt" or total_cnt	INTEGER or BIGINT	Total count of observations.

NaiveBayesPredict Output

Output Table Schema

Each row of the table represents one observation.

Column	Data Type	Description
id	INTEGER	Row (observation) identifier.
prediction	VARCHAR	Prediction for observation.
loglik_response_i	DOUBLE PRECISION	[Column appears once for each specified <i>response</i> .] Loglikelihood (natural logarithm of probability) that observation has <i>response</i> .

NaiveBayesPredict Example

Input

- Input table: nb_iris_input_test
- Model: nb_iris_model

The model is created in the Naive Bayes example in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Input Table Column Descriptions

Column	Description
id	Unique identifier of observation
sepal_length	Numeric

Column	Description
sepal_width	Numeric
petal_length	Numeric
petal_width	Numeric
species	Setosa, versicolor, or virginica

nb_iris_input_test

id	sepal_length	sepal_width	petal_length	petal_width	species
5	5	3.6	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
15	5.8	4	1.2	0.2	setosa
20	5.1	3.8	1.5	0.3	setosa
25	4.8	3.4	1.9	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
35	4.9	3.1	1.5	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
45	5.1	3.8	1.9	0.4	setosa
50	5	3.3	1.4	0.2	setosa
55	6.5	2.8	4.6	1.5	versicolor
60	5.2	2.7	3.9	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
70	5.6	2.5	3.9	1.1	versicolor
75	6.4	2.9	4.3	1.3	versicolor
80	5.7	2.6	3.5	1	versicolor
85	5.4	3	4.5	1.5	versicolor
90	5.5	2.5	4	1.3	versicolor
95	5.6	2.7	4.2	1.3	versicolor
100	5.7	2.8	4.1	1.3	versicolor
105	6.5	3	5.8	2.2	virginica
110	7.2	3.6	6.1	2.5	virginica

id	sepal_length	sepal_width	petal_length	petal_width	species
115	5.8	2.8	5.1	2.4	virginica
120	6	2.2	5	1.5	virginica
125	6.7	3.3	5.7	2.1	virginica
130	7.2	3	5.8	1.6	virginica
135	6.1	2.6	5.6	1.4	virginica
140	6.9	3.1	5.4	2.1	virginica
145	6.7	3.3	5.7	2.5	virginica
150	5.9	3	5.1	1.8	virginica

nb_iris_model

class	variable	type	category	cnt	sum	sumSq	totalcnt
setosa	sepal_width	NUMERIC		40	136.700000524521	473.290003499985	40
setosa	petal_width	NUMERIC		40	10.1000002026558	3.03000012755394	40
setosa	sepal_length	NUMERIC		40	199.900000095367	1004.270000005722	40
setosa	petal_length	NUMERIC		40	57.6999998092651	84.2099996709824	40
versicolor	sepal_width	NUMERIC		40	111.10000038147	313.130002088547	40
versicolor	petal_width	NUMERIC		40	53.299999833107	72.7099995040894	40
versicolor	sepal_length	NUMERIC		40	239.599999427795	1446.13999296188	40
versicolor	petal_length	NUMERIC		40	172.399999141693	752.219992570878	40
virginica	sepal_width	NUMERIC		40	118.799999952316	356.539999780655	40
virginica	petal_width	NUMERIC		40	81.1999989748001	166.999995970726	40
virginica	sepal_length	NUMERIC		40	264.400000572205	1764.92000530243	40
virginica	petal_length	NUMERIC		40	222.299999713898	1249.1499958992	40

SQL Call

```
DROP TABLE nb_iris_predict;
```

```
CREATE MULTISET TABLE nb_iris_predict AS (
  SELECT * FROM NaiveBayesPredict (
    ON nb_iris_input_test PARTITION BY ANY
    ON nb_iris_model AS Model DIMENSION
    USING
    IDColumn ('id')
    NumericInputs ('sepal_length','sepal_width','petal_length','petal_width')
```

```

    Responses ('virginica','setosa','versicolor')
  ) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM nb_iris_predict ORDER BY 1;
```

The output provides a prediction for each row in the test data set and specifies the log likelihood values that were used to make the predictions for each category.

id	prediction	loglik_virginica	loglik_setosa	loglik_versicolor
5	setosa	-60.9907330174083	0.940424559067427	-38.2319825308929
10	setosa	-61.5861966261907	-0.173043897170957	-37.6660830556247
15	setosa	-64.7169548001753	-3.55476375390931	-42.613272284101
20	setosa	-57.7992844148636	0.531796840642284	-35.7613053354934
25	setosa	-55.0939143017897	-3.23703029869347	-32.1179858509341
30	setosa	-58.0673073752287	0.109611164911179	-34.9285997859276
35	setosa	-58.1980267787658	0.660202577013632	-34.9335988704833
40	setosa	-58.3538858459019	0.976840811041703	-35.4425587940391
45	setosa	-50.3847602463201	-4.36921429673761	-29.0537478266948
50	setosa	-59.4745348026195	1.00257959230347	-36.5026022674224
55	versicolor	-5.22108005914589	-270.465431908161	-1.7396367893394
60	versicolor	-11.3356467465064	-174.565470791378	-2.31925264962004
65	versicolor	-12.6496488706934	-138.435722453706	-2.1898005756116
70	versicolor	-15.236843619572	-152.47255627778	-2.3538459106499
75	versicolor	-8.34632493685681	-214.383653794905	-1.14727508911532
80	versicolor	-18.455946984498	-109.900955754698	-3.72743011721095
85	versicolor	-7.00283150694931	-249.656488976769	-2.00455589365379
90	versicolor	-12.0279925543069	-177.470336291088	-1.74539749109463
95	versicolor	-10.1802450220293	-198.037109900803	-1.10567314638237
100	versicolor	-10.1315405651018	-187.294956922171	-1.02885306444447
105	virginica	-1.58321671192447	-540.56351949849	-14.859643718252

id	prediction	loglik_virginica	loglik_setosa	loglik_versicolor
110	virginica	-6.11301966870239	-654.801984259278	-28.8385135092999
115	virginica	-3.64635253153959	-456.647579953406	-15.3298808321577
120	versicolor	-7.73615017754911	-322.909009762056	-3.53629430321742
125	virginica	-1.87627054598219	-509.817023097936	-13.7515396871732
130	virginica	-3.36908052149115	-469.802937074554	-9.13832860900173
135	versicolor	-5.81482980902253	-403.678170868448	-4.51644862072851
140	virginica	-1.48430911768034	-463.610989255182	-12.0238603485835
145	virginica	-3.82266629516761	-576.395460020916	-22.6942168473031
150	virginica	-2.57004648415525	-366.506113945482	-4.84887216455807

NaiveBayesTextClassifierPredict

The NaiveBayesTextClassifierPredict function uses the model table output by ML Engine NaiveBayesTextClassifierTrainer function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

NaiveBayesTextClassifierPredict Syntax

```
SELECT * FROM NaiveBayesTextClassifierPredict (
  ON { table | view | (query) } AS PredictorValues PARTITION BY doc_id_column
  ON { table | view | (query) } AS Model DIMENSION
  USING
    InputTokenColumn ('token_column')
    [ ModelType ({ 'Multinomial' | 'Bernoulli' }) ]
    DocIDColumns ('doc_id_column'[,...])
    [ ModelTokenColumn ('model_token_column')
      ModelCategoryColumn ('model_category_column')
      ModelProbColumn ('model_token_count_column') ]
    [ TopK ({ num_of_top_k_predictions | 'num_of_top_k_predictions' }) ]
) AS alias;
```

NaiveBayesTextClassifierPredict Syntax Elements

InputTokenColumn

Specify the name of the PredictorValues column that contains the tokens.

ModelType

[Optional] Specify the model type of the text classifier.

Default: 'Multinomial'

DocIDColumns

Specify the names of the PredictorValues columns that contain the document identifier.

ModelTokenColumn

[Optional] Specify the name of the Model table column that contains the tokens.

Default: First column of Model table

ModelCategoryColumn

[Optional] Specify the name of the Model table column that contains the prediction categories.

Default: Second column of Model table

ModelProbColumn

[Optional] Specify the name of the Model table column that contains the token counts.

Default: Third column of Model table

TopK

[Optional] Specify the number of most likely prediction categories to output with their loglikelihood values (for example, the top 10 most likely prediction categories).

Default: All prediction categories

Note:

Specify either all or none of the syntax elements ModelTokenColumn, ModelCategoryColumn, and ModelProbColumn.

NaiveBayesTextClassifierPredict Input

Table	Description
PredictorValues	Contains test data, for which to predict outcomes, in document-token pairs. To transform input document into this form, input it to ML Engine function TextTokenizer or TextParser. Note: TextTokenizer and TextParser have language-processing limitations that might limit support for Unicode input data (see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003).
Model	Model output by ML Engine NaiveBayesTextClassifierTrainer function. For schema, see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003.

PredictorValues Schema

Column	Data Type	Description
<i>doc_id_column</i>	CHARACTER, VARCHAR, INTEGER, or SMALLINT	[Column appears once for each specified <i>doc_id_column</i> .] Identifier of document that contains classified training tokens.
<i>token_column</i>	CHARACTER or VARCHAR	Classified training token.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
token	CHARACTER or VARCHAR	Classified training token.
category	CHARACTER or VARCHAR	Prediction category for token.
prob	DOUBLE PRECISION	Probability that token is in category.

NaiveBayesTextClassifierPredict Output**Output Table Schema**

Column	Data Type	Description
doc_id	CHARACTER, VARCHAR, INTEGER, or SMALLINT	Single- or multiple-column document identifier.
prediction	VARCHAR	Prediction category.
loglik	DOUBLE PRECISION	Loglikelihood that document belongs to category.

NaiveBayesTextClassifierPredict Example**Input**

The input table, `complaints_test`, is a log of vehicle complaints. The example applies ML Engine TextTokenizer function to `complaints_test` to create a table of test data, `complaints_tokens_test`, and uses the model `complaints_tokens_model`, output by ML Engine NaiveBayesTextClassifierTrainer function.

complaints_test

doc_id	text_data
1	ELECTRICAL CONTROL MODULE IS SHORTENING OUT, CAUSING THE VEHICLE TO STALL. ENGINE WILL BECOME TOTALLY INOPERATIVE. CONSUMER HAD TO CHANGE

doc_id	text_data
	ALTERNATOR/ BATTERY AND STARTER, AND MODULE REPLACED 4 TIMES, BUT DEFECT STILL OCCURRING CANNOT DETERMINE WHAT IS CAUSING THE PROBLEM.
2	ABS BRAKES FAIL TO OPERATE PROPERLY, AND AIR BAGS FAILED TO DEPLOY DURING A CRASH AT APPROX. 28 MPH IMPACT. MANUFACTURER NOTIFIED.
3	WHILE DRIVING AT 60 MPH GAS PEDAL GOT STUCK DUE TO THE RUBBER THAT IS AROUND THE GAS PEDAL.
4	THERE IS A KNOCKING NOISE COMING FROM THE CATALYTIC CONVERTER ,AND THE VEHICLE IS STALLING. ALSO, HAS PROBLEM WITH THE STEERING.
5	CONSUMER WAS MAKING A TURN ,DRIVING AT APPROX 5- 10 MPH WHEN CONSUMER HIT ANOTHER VEHICLE. UPON IMPACT, DUAL AIRBAGS DID NOT DEPLOY . ALL DAMAGE WAS DONE FROM ENGINE TO TRANSMISSION,TO THE FRONT OF VEHICLE, AND THE VEHICLE CONSIDERED A TOTAL LOSS.
6	WHEEL BEARING AND HUBS CRACKED, CAUSING THE METAL TO GRIND WHEN MAKING A RIGHT TURN. ALSO WHEN APPLYING THE BRAKES, PEDAL GOES TO THE FLOOR, CAUSE UNKNOWN. WAS ADVISED BY MIDAS NOT TO DRIVE VEHICLE- WHEEL COULD COME OFF.
7	DRIVING ABOUT 5-10 MPH, THE VEHICLE HAD A LOW FRONTAL IMPACT IN WHICH THE OTHER VEHICLE HAD NO DAMAGES. UPON IMPACT, DRIVER'S AND THE PASSENGER'S AIR BAGS DID NOT DEPLOY, RESULTING IN INJURIES. PLEASE PROVIDE FURTHER INFORMATION AND VIN#.
8	THE AIR BAG WARNING LIGHT HAS COME ON. INDICATING AIRBAGS ARE INOPERATIVE. THEY WERE FIXED ONE AT THE TIME, BUT PROBLEM HAS REOCCURRED.
9	CONSUMER WAS DRIVING WEST WHEN THE OTHER CAR WAS GOING EAST. THE OTHER CAR TURNED IN FRONT OF CONSUMER'S VEHICLE, CONSUMER HIT OTHER VEHICLE AND STARTED TO SPIN AROUND ,COULDN'T STOP, RESULTING IN A CRASH. UPON IMPACT, AIRBAGS DIDN'T DEPLOY.
10	WHILE DRIVING ABOUT 65 MPH AND THE TRANSMISSION MADE A STRANGE NOISE, AND THE LEFT FRONT AXLE LOCKED UP. THE DEALER HAS REPAIRED THE VEHICLE.

SQL Calls

Call to TextTokenizer to create complaints_tokens_test:

```
CREATE MULTISET TABLE complaints_tokens_test AS (
  SELECT doc_id, lower(cast(token AS VARCHAR(1024))) AS token
  FROM TextTokenizer (
    ON complaints_test PARTITION BY ANY
    USING
    TextColumn ('text_data')
    OutputByWord ('true')
    Accumulate ('doc_id')
```

```

    ) AS dt
  ) WITH DATA;

```

Call to NaiveBayesTextClassifierPredict:

```

SELECT * FROM NaiveBayesTextClassifierPredict (
  ON complaints_tokens_test AS PredictorValues PARTITION BY doc_id
  ON complaints_tokens_model AS Model DIMENSION
  USING
    InputTokenColumn ('token')
    ModelType ('Bernoulli')
    DocIDColumns ('doc_id')
    TopK ('1')
  ) AS dt ORDER BY doc_id;

```

Output

doc_id	prediction	loglik
1	no_crash	-98.5428474553942
2	no_crash	-93.588731591964
3	no_crash	-74.6281619901653
4	no_crash	-80.2178104775681
5	crash	-115.803709744721
6	no_crash	-116.281532818164
7	crash	-111.174594434561
8	no_crash	-92.1427644980375
9	no_crash	-109.322164262443
10	no_crash	-82.820437621875

NGramSplitter

The NGramSplitter function *tokenizes* (splits) an input stream of text and outputs *n* multigrams (called *n-grams*) based on the specified Reset, Punctuation, and Delimiter syntax elements. NGramSplitter first splits sentences, next removes punctuation characters from them, and finally splits the words into *n*-grams.

NGramSplitter provides more flexibility than standard tokenization when performing text analysis. Many two-word phrases carry important meaning (for example, "machine learning") that single-word tokens do not capture. This, combined with additional analytical techniques, can be useful for performing sentiment analysis, topic identification, and document classification.

NGramSplitter considers each input row to be one document, and returns a row for each unique n -gram in each document. NGramSplitter also returns, for each document, the counts of each n -gram and the total number of n -grams.

Note:

- This function requires the UTF8 client character set.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

NGramSplitter Syntax

```
SELECT * FROM NGramSplitter (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')
  Grams ({ gram_number | 'value_range' }[,...])
  [ OverLapping({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ ConvertToLowerCase ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ Reset ('reset_character...') ]
  [ Punctuation ('punctuation_character...') ]
  [ Delimiter ('delimiter_character...') ]
  [ OutputTotalGramCount ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ TotalCountColName ('total_count_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ NGramColName ('ngram_column') ]
  [ GramLengthColName ('gram_length_column') ]
  [ FrequencyColName ('frequency_column') ]
) AS alias;
```

NGramSplitter Syntax Elements

TextColumn

Specify the name of the column that contains the input text. This column must have a SQL string data type.

Grams

Specify the length, in words, of each n -gram (that is, the value of n). A *value_range* has the syntax *integer1-integer2*, where *integer1* ≤ *integer2*. The values of n , *integer1*, and *integer2* must be positive.

OverLapping

[Optional] Specify whether the function allows overlapping n -grams.

Default: 'true' (Each word in each sentence starts an n -gram, if enough words follow it in the same sentence to form a whole n -gram of the specified size. For information on sentences, see the Reset syntax element description.)

ConvertToLowerCase

[Optional] Specify whether the function converts all letters in the input text to lowercase.

Default: 'true'

Reset

[Optional] Specify, in a string, the characters that can end a sentence. At the end of a sentence, the function discards any partial n -grams and searches for the next n -gram at the beginning of the next sentence. An n -gram cannot span sentences.

Default: ' . , ? ! '

Punctuation

[Optional] Specify, in a string, the punctuation characters for the function to remove before evaluating the input text.

Note:

Punctuation characters can be from both Unicode and Latin character sets.

Default: '`~#^&*()- '

Delimiter

[Optional] Specify the character or string that separates words in the input text.

Default: ' ' (space)

OutputTotalGramCount

[Optional] Specify whether the function returns the total number of n -grams in the document (that is, in the row) for each length n specified in the Grams syntax element. If you specify 'true', the TotalCountColName syntax element determines the name of the output table column that contains these totals.

Note:

The total number of n -grams is not necessarily the number of unique n -grams.

Default: 'false'

TotalCountColName

[Optional] Specify the name of the output table column that appears if the value of the OutputTotalGramCount syntax element is 'true'.

Default: 'totalcnt'

Accumulate

[Optional] Specify the names of the input table columns to copy to the output table for each n -gram. These columns cannot have the same names as those specified by the syntax elements NGramColName, GramLengthColName, and TotalCountColName.

Default: All input columns for each n -gram

NGramColName

[Optional] Specify the name of the output table column that is to contain the created n -grams.

Default: 'ngram'

GramLengthColName

[Optional] Specify the name of the output table column that is to contain the length of n -gram (in words).

Default: 'n'

FrequencyColName

[Optional] Specify the name of the output table column that is to contain the count of each unique n -gram (that is, the number of times that each unique n -gram appears in the document).

Default: 'frequency'

NGramSplitter Input

Input Table Schema

Each row of the table has a document to tokenize. The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>text_column</i>	VARCHAR or CLOB	Document to tokenize.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

NGramSplitter Output

Output Table Schema

The table has a row for each unique n -gram in each input document.

Column	Data Type	Description
<i>total_count_column</i>	INTEGER	[Column appears only with TotalCountColName ('true').] Total number of n -grams in document for each length n specified in Grams syntax element.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.

Column	Data Type	Description
<i>ngram_column</i>	VARCHAR	Created <i>n</i> -gram.
<i>gram_length_column</i>	INTEGER	Length of <i>n</i> -gram in words (value <i>n</i>).
<i>frequency_column</i>	INTEGER	Count of each unique <i>n</i> -gram in document.

NGramSplitter Example

Input

The input table, `paragraphs_input`, contains sentences about commonly used machine learning techniques.

`paragraphs_input`

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of <i>n</i> points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible
...

SQL Call

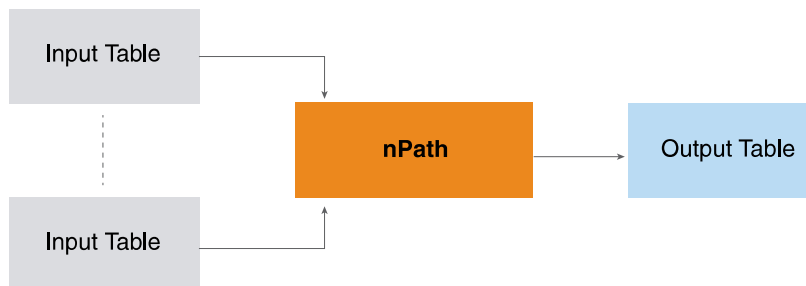
```
SELECT * FROM NGramSplitter (
  ON paragraphs_input
  USING
    TextColumn ('paratext')
    Grams ('4-6')
    OutputTotalGramCount ('true')
) AS dt;
```

Output

paraid	paratopic	ngram	n	frequency	totalcnt
1	Decision Trees	decision tree learning uses	4	1	73
1	Decision Trees	decision tree learning uses a	5	1	66
1	Decision Trees	decision tree learning uses a decision	6	1	60
1	Decision Trees	tree learning uses a	4	1	73
1	Decision Trees	tree learning uses a decision	5	1	66
1	Decision Trees	tree learning uses a decision tree	6	1	60
1	Decision Trees	learning uses a decision	4	1	73
1	Decision Trees	learning uses a decision tree	5	1	66
1	Decision Trees	learning uses a decision tree as	6	1	60
...

nPath®

The nPath function scans a set of rows, looking for patterns that you specify. For each set of input rows that matches the pattern, nPath produces a single output row. The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.

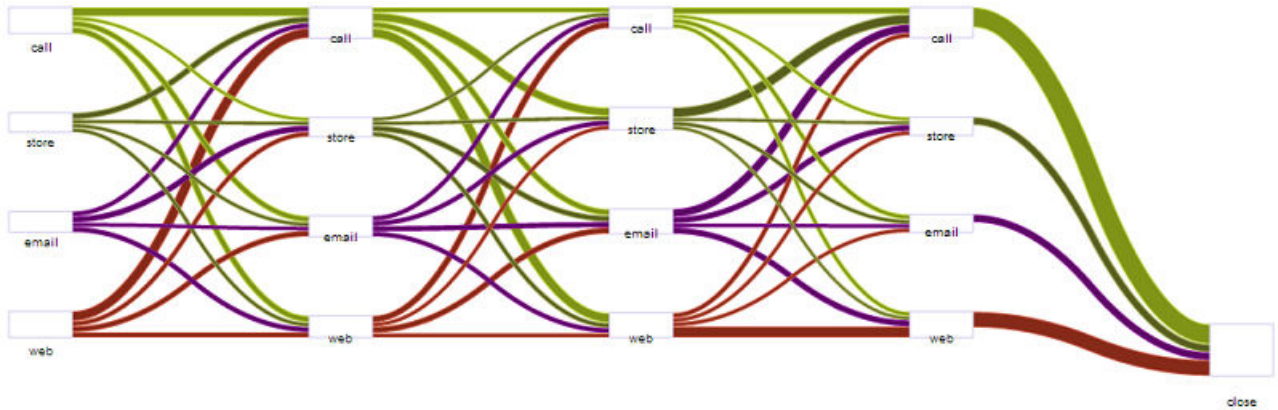


nPath is useful when your goal is to identify the paths that lead to an outcome. For example, you can use nPath to analyze:

- Web site click data, to identify paths that lead to sales over a specified amount
- Sensor data from industrial processes, to identify paths to poor product quality
- Healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- Financial data for individuals, to identify paths that provide information about credit or fraud risks

The output from the nPath function can be input to other ML Engine functions or to a visualization tool such as Teradata® AppCenter.

Sankey Diagram of NewSQL Engine nPath Output



An nPath call specifies:

- [Mode \(overlapping or nonoverlapping\)](#)
- [Pattern to match](#)
- [Symbols to use](#)
- [Optional] [Filters to apply](#)
- [Results to output](#)

Note:

nPath does not support Unicode.

nPath Syntax

```
SELECT * FROM nPath (
  ON { table | view | (query) }
  PARTITION BY partition_column
  ORDER BY order_column [ ASC | DESC ][...]
  [ ON { table | view | (query) }
    [ PARTITION BY partition_column | DIMENSION ]
    ORDER BY order_column [ ASC | DESC ]
  ][...]
  USING
  Mode ({ OVERLAPPING | NONOVERLAPPING })
  Pattern ('pattern')
  Symbols ({ col_expr = symbol_predicate AS symbol}[,...])
  [ Filter (filter_expression[,...]) ]
```

```
Result ({ aggregate_function (expression OF [ANY] symbol [,...]) AS alias_1 }[,...])
) AS alias_2;
```

nPath Syntax Elements

Mode

Specify the pattern-matching mode:

Option	Description
OVERLAPPING	Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern.
NONOVERLAPPING	Start next pattern search at row that follows last pattern match.

Pattern

Specify the pattern for which the function searches. You compose *pattern* with the symbols (which you define in the Symbols syntax element), operators, and parentheses.

When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses. For more information, see [nPath Patterns](#).

Symbols

Defines the symbols that appear in the values of the Pattern and Result syntax elements. The *col_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol_predicate* is a SQL predicate (often a column name).

For example, this Symbols syntax element is for analyzing website visits:

```
Symbols (
  pagetype = 'homepage' AS H,
  pagetype <> 'homepage' AND pagetype <> 'checkout' AS PP,
  pagetype = 'checkout' AS CO
)
```

The *symbol* is case-insensitive; however, a *symbol* of one or two uppercase letters is easy to identify in patterns.

If *col_expr* represents a column that appears in multiple input tables, you must qualify the ambiguous column name with its table name. For example:

```
Symbols (
  weblog.pagetype = 'homepage' AS H,
  weblog.pagetype = 'thankyou' AS T,
  ads.adname = 'xmaspromo' AS X,
  ads.adname = 'realtorpromo' AS R
)
```

For more information about symbols that appear in the Pattern syntax element value, see [nPath Symbols](#). For more information about symbols that appear in the Result syntax element value, see [nPath Results](#).

Filter

[Optional] Specify filters to impose on the matched rows. The function combines the filter expressions using the AND operator.

This is the *filter_expression* syntax:

```
symbol_expression comparison_operator symbol_expression
```

The two symbol expressions must be type-compatible. This is the *symbol_expression* syntax:

```
{ FIRST | LAST }(column_with_expression OF [ANY](symbol[, ...]))
```

The *column_with_expression* cannot contain the operator AND or OR, and all its columns must come from the same input. If the function has multiple inputs, *column_with_expression* and *symbol* must come from the same input.

The *comparison_operator* is either <, >, <=, >=, =, or <>.

Result

Defines the output columns. The *col_expr* is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies *aggregate_function* to these values. For details, see [nPath Results](#).

The function evaluates this syntax element once for every matched pattern in the partition (that is, it outputs one row for each pattern match).

nPath Input

The function requires at least one partitioned input table, and can have additional input tables that are either partitioned or DIMENSION tables.

Note:

If the input to nPath is nondeterministic, the results are nondeterministic.

Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	INTEGER or VARCHAR	Column by which every partitioned input table is partitioned.
<i>order_column</i>	INTEGER or VARCHAR	Column by which every input table is ordered.
<i>input_column</i>	INTEGER or VARCHAR	Contains data to search for patterns.

nPath Output

The Result syntax element determines the output—see [nPath Results](#).

nPath Symbols

A *symbol* identifies a row in the Pattern and Result syntax elements. A symbol can be any valid identifier (that is, a sequence of characters and digits that begins with a character) but is typically one or two uppercase letters. Symbols are case-insensitive; that is, 'SU' is identical to 'su', and the system reports an error if you use both.

For example, suppose that you have this input table:

record	city	temp	rh	cloudcover	windspeed	winddirection	rained_next_day
1		81	30	0.0	5	NW	1
2	Tempe	76	40	0.2	15	NE	0
3		70	70	0.4	10	N	0
4	Tusayan	75	50	0.4	5	NW	0

This table has examples of symbol definitions and the rows of the table that they match in NONOVERLAPPING mode:

Symbol Definition	Rows Matched						
temp >= 80 AS H	1						
winddirection = 'NW' AS NW	1, 4						
winddirection = 'NW' OR windspeed > 12 AS W	1, 2, 4						
cloudcover <> 0.0 AND rh > 35 AS C	2, 3, 4						
TRUE AS A	1, 2, 3, 4 This symbol definition matches all rows, for any input table.						
city like 'tu%' AS TU	The like operator depends on Teradata Session mode: <table> <tr> <th>Mode</th><th>Match</th></tr> <tr> <td>BTET</td><td>1, 3, 4</td></tr> <tr> <td>ANSI</td><td>None</td></tr> </table>	Mode	Match	BTET	1, 3, 4	ANSI	None
Mode	Match						
BTET	1, 3, 4						
ANSI	None						
city not like 'tu%' AS TU	2						
city not like 'Tu%' AS N	2						
city like 'Tu%n' as T	1, 3, 4						

Symbol Definition	Rows Matched
	The % operator matches any number of characters.
city like 'Tu__n' as T	1, 3 The underscore (_) operator matches any single character. The pattern 'Tu__n' has three underscores, so it matches 'Tucson' but not 'Tusayan'.

Rows with NULL values do not match any symbol. That is, the function ignores rows with missing values.

LAG and LEAD Expressions in Symbol Predicates

You can create symbol predicates that compare a row to a previous or subsequent row, using a LAG or LEAD operator.

LAG Expression Syntax

```
{ current_expr operator LAG (previous_expr, lag_rows [, default]) |
  LAG (previous_expr, lag_rows [, default]) operator current_expr }
```

where:

- *current_expr* is the name of a column from the current row (or an expression operating on this column).
- *operator* is either >, >=, <, <=, =, or <>
- *previous_expr* is the name of a column from a previous row (or an expression operating on this column).
- *lag_rows* is the number of rows to count backward from the current row to reach the previous row. For example, if *lag_rows* is 1, the previous row is the immediately preceding row.
- *default* is the value to use for *previous_expr* when there is no previous row (that is, when the current row is the first row or there is no row that is *lag_rows* before the current row).

LAG and LEAD Expression Rules

- A symbol definition can have multiple LAG and LEAD expressions.
- A symbol definition that has a LAG or LEAD expression cannot have an OR operator.
- If a symbol definition has a LAG or LEAD expression and the input is not a table, you must create an alias of the input query, as in [LAG and LEAD Expressions Example: Alias for Input Query](#).

LAG and LEAD Expressions Example: Alias for Input Query

Input**bank_web_clicks**

customer_id	session_id	page	datestamp
529	0	ACCOUNT SUMMARY	2004-03-17 16:35:00
529	0	FAQ	2004-03-17 16:38:00
529	0	ACCOUNT HISTORY	2004-03-17 16:42:00
529	0	FUNDS TRANSFER	2004-03-17 16:45:00
529	0	ONLINE STATEMENT ENROLLMENT	2004-03-17 16:49:00
529	0	PROFILE UPDATE	2004-03-17 16:50:00
529	0	ACCOUNT SUMMARY	2004-03-17 16:51:00
529	0	CUSTOMER SUPPORT	2004-03-17 16:53:00
529	0	VIEW DEPOSIT DETAILS	2004-03-17 16:57:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:16:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:18:00
529	1	FAQ	2004-03-18 01:20:00
...

SQL Call

```

SELECT * FROM nPath (
  ON (SELECT customer_id, session_id, datestamp, page FROM bank_web_clicks) AS
  dt1
  PARTITION BY customer_id, session_id
  ORDER BY datestamp
  USING
  Mode (NONOVERLAPPING)
  Pattern ('(DUP|A)*')
  Symbols (
    TRUE AS A,
    page = LAG (page,1) AS DUP
  )
  Result (
    FIRST (customer_id OF any (A)) AS customer_id,
    FIRST (session_id OF A) AS session_id,
    FIRST (datestamp OF A) AS first_date,
    LAST (datestamp OF ANY(A,DUP)) AS last_date,
    ACCUMULATE (page OF A) AS page_path,

```



```

    ACCUMULATE (page of DUP) AS dup_path
  )
) AS dt2;

```

Output

Columns 1-4

customer_id	session_id	first_date	last_date
529	0	2004-03-17 16:35:00	2004-03-17 16:57:00
529	1	2004-03-18 01:16:00	2004-03-18 01:28:00
529	2	2004-03-18 09:22:00	2004-03-18 09:36:00
529	3	2004-03-18 22:41:00	2004-03-18 22:55:00
529	4	2004-03-19 08:33:00	2004-03-19 08:41:00
529	5	2004-03-19 10:06:00	2004-03-19 10:14:00
...

Columns 5-6

page_path	dup_path
[ACCOUNT SUMMARY, FAQ, ACCOUNT HISTORY, FUNDS TRANSFER, ONLINE STATEMENT ENROLLMENT, PROFILE UPDATE, ACCOUNT SUMMARY, CUSTOMER SUPPORT, VIEW DEPOSIT DETAILS]	[]
[ACCOUNT SUMMARY, FAQ, ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, ACCOUNT SUMMARY, ACCOUNT HISTORY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, FUNDS TRANSFER, ACCOUNT SUMMARY, FAQ]	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY, ACCOUNT HISTORY, FAQ, ACCOUNT SUMMARY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, FAQ, VIEW DEPOSIT DETAILS, FAQ]	[]
[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	[VIEW DEPOSIT DETAILS]
...	...

LAG and LEAD Expressions Example: No Alias for Input Query

Input**aggregate_clicks**

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250
1040	2	Appliances	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16 11:18:16	Company3	400
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...

SQL Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime ASC
  USING
```

```

Mode (NONOVERLAPPING)
Pattern ('H+.D*.X*.P1.P2+')
Symbols (
  TRUE AS X,
  pagetype = 'home' AS H,
  pagetype <> 'home' AND pagetype <> 'checkout' AS D,
  pagetype = 'checkout' AS P1,
  pagetype = 'checkout' AND
  productprice > 100 AND
  productprice > LAG (productprice, 1, 100) AS P2
)
Result (
  FIRST (productname OF P1) AS first_product,
  MAX_CHOOSE (productprice, productname OF P2) AS max_product,
  FIRST (sessionid OF P2) AS sessionid
)
) AS dt ORDER BY sessionid;

```

Output

first_product	max_product	sessionid
bookcases	cellphones	5

nPath Patterns

The value of the Pattern syntax element specifies the sequence of rows for which the function searches. You compose the pattern definition, *pattern*, with symbols (which you define in the Symbols syntax element), operators, and parentheses. In the pattern definition, symbols represent rows. You can combine symbols with pattern operators to define simple or complex patterns of rows for which to search.

The following table lists and describes the basic pattern operators, in decreasing order of precedence. In the table, A and B are symbols that have been defined in the Symbols syntax element.

Basic Pattern Operators

Operator	Description	Precedence
A	Matches one row that meets the definition of A.	1 (highest)
A.	Matches one row that meets the definition of A.	1
A?	Matches 0 or 1 rows that satisfy the definition of A.	1
A*	Matches 0 or more rows that satisfy the definition of A (greedy operator).	1
A+	Matches 1 or more rows that satisfy the definition of A (greedy operator).	1

Operator	Description	Precedence
<code>A.B</code>	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B.	2
<code>A B</code>	Matches one row that meets the definition of either A or B.	3

The nPath function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators. For more information, see [nPath Greedy Pattern Matching](#).

These examples show the pattern operator precedence rules:

- `A.B+` is the same as `A.(B+)`
- `A|B*` is the same as `A|(B*)`
- `A.B|C` is the same as `(A.B)|C`

Example:

```
A.(B|C)+.D?.X*.A
```

The preceding pattern definition matches any set of rows whose first row meets the definition of symbol A, followed by a nonempty sequence of rows, each of which meets the definition of either symbol B or C, optionally followed by one row that meets the definition of symbol D, followed by any number of rows that meet the definition of symbol X, and ending with a row that meets the definition of symbol A.

You can use parentheses to define precedence rules. Parentheses are recommended for clarity, even where not strictly required.

To indicate that a sequence of rows must start or end with a row that matches a certain symbol, use the start anchor (^) or end anchor (\$) operator.

Start Anchor and End Anchor Pattern Operators

Operator	Description
<code>^A</code>	Appears only at the beginning of a pattern. Indicates that a set of rows must start with a row that meets the definition of A.
<code>A\$</code>	Appears only at the end of a pattern. Indicates that a set of rows must end with a row that meets the definition of A.

Subpattern operators let you specify how often a subpattern must appear in a match. You can specify a minimum number, exact number, or range. In the following table, X represents any pattern definition composed of symbols and any of the previously described pattern operators.

Subpattern Operators

Operator	Description
<code>(X){a}</code>	Matches exactly a occurrences of the pattern X.

Operator	Description
<code>(X){a,}</code>	Matches at least <code>a</code> occurrences of the pattern <code>X</code> .
<code>(X){a,b}</code>	Matches at least <code>a</code> and no more than <code>b</code> occurrences of the pattern <code>X</code> .

nPath Greedy Pattern Matching

The `nPath` function uses greedy pattern matching, finding the longest available match despite any nongreedy operators in the pattern.

For example, consider the input table `link2`:

nPath Greedy Pattern Matching Examples Input Table link2

userid	title	startdate	enddate
21	Chief Exec Officer	1994-10-01	2005-02-28
21	Software Engineer	1996-10-01	2001-06-30
21	Software Engineer	1998-10-01	2001-06-30
21	Chief Exec Officer	2005-03-01	2007-03-31
21	Chief Exec Officer	2007-06-01	null

The following query returns the following table:

```
SELECT job_transition_path, count(*) AS path_count FROM nPath (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*')
  Symbols (
    job_title like '%Software Eng%' AS ENGR,
    TRUE AS OTHER,
    job_title like 'Chief Exec Officer' AS CEO
  )
  Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;
```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, `CEO` matches the first row, `ENGR` matches the second row, and `OTHER*` matches the remaining rows:

title	pattern('CEO.ENGR.OTHER*')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

The following query returns the following table:

```

SELECT job_transition_path , count(*) AS path_count FROM nPath (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*.CEO')
  Symbols (
    job_title like '%Software Eng%' AS ENGR,
    TRUE AS OTHER,
    job_title like 'Chief Exec Officer' AS CEO
  )
  Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;

```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, OTHER* matches the next two rows, and CEO matches the last row:

title	pattern('CEO.ENGR.OTHER*.CEO')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

nPath Filters

The Filter syntax element specifies filters to impose on the matched rows.

nPath Filters Example

Using clickstream data from an online store, this example finds the sessions where the user visited the checkout page within 10 minutes of visiting the home page. Because there is no way to know in advance how many rows might appear between the home page and the checkout page, the example cannot use a LAG or LEAD expression. Therefore, it uses the Filter syntax element.

Input

clickstream

userid	sessionid	clicktime	pagetype
1	1	10-10-2012 10:15	home
1	1	10-10-2012 10:16	view
1	1	10-10-2012 10:17	view
1	1	10-10-2012 10:20	checkout
1	1	10-10-2012 10:30	checkout
1	1	10-10-2012 10:35	view
1	1	10-10-2012 10:45	view
2	2	10-10-2012 13:15	home
2	2	10-10-2012 13:16	view
2	2	10-10-2012 13:43	checkout
2	2	10-10-2012 13:35	view
2	2	10-10-2012 13:45	view

SQL Call

```
SELECT * FROM Npath (
  ON clickstream PARTITION BY userid ORDER BY clicktime
  USING
  Symbols (
    pagetype='home' AS home,
    pagetype <> 'home' AND pagetype <> 'checkout' AS clickview,
    pagetype='checkout' AS checkout
  )
  Pattern ('home.clickview*.checkout')
  Result (
    FIRST(userid of ANY(home, checkout, clickview)) AS userid,
    FIRST (sessionid of ANY(home, checkout, clickview)) AS sessionid,
    COUNT (*) of any(home, checkout, clickview) AS cnt,
    FIRST (clicktime of ANY(home)) AS firsthome,
    LAST (clicktime of ANY(checkout)) AS lastcheckout
```

```

)
Filter (
  FIRST (clicktime + interval '10' minute OF ANY (home)) >
  FIRST (clicktime of any(checkout))
)
Mode (NONOVERLAPPING)
);

```

Output

userid	sessionid	cnt	firsthome	lastcheckout
1	1	4	2012-10-10 10:15:00	2012-10-10 10:20:00

nPath Results

The Result syntax element defines the output columns, specifying the values to retrieve from the matched rows and the aggregate function to apply to these values.

For each pattern, the nPath function can apply one or more aggregate functions to the matched rows and output the aggregated results. These are the supported aggregate functions:

- SQL aggregate functions AVG, COUNT, MAX, MIN, and SUM, described in *Teradata Vantage™ SQL Functions, Expressions, and Predicates*, B035-1145
- ML Engine nPath sequence aggregate functions described in the following table

In the following table, *col_expr* is an expression whose value is a column name, *symbol* is defined by the Symbols syntax element, and *symbol_list* has this syntax:

```
{ symbol | ANY (symbol[,...]) }
```

Function	Description
COUNT ({ * [DISTINCT] col_expr } OF symbol_list)	Returns either the number of total number of matched rows (*) or the number (or distinct number) of col_expr values in the matched rows.
FIRST (col_expr OF symbol_list)	Returns the col_expr value of the first matched row.
LAST (col_expr OF symbol_list)	Returns the col_expr value of the last matched row.
NTH (col_expr, n OF symbol_list)	Returns the col_expr value of the nth matched row, where n is a nonzero value of the data type SMALLINT, INTEGER, or BIGINT. The sign of n determines whether the nth matched row is nth from the first or last matched row. For example, if n is 1, the nth matched row is

Function	Description
	the first matched row, and if <i>n</i> is -1, the <i>n</i> th matched row is the last matched row. If <i>n</i> is greater than the number of matched rows, the <i>n</i> th function returns NULL.
FIRST_NOTNULL (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the first non-null <i>col_expr</i> value in the matched rows.
LAST_NOTNULL (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the last non-null <i>col_expr</i> value in the matched rows.
MAX_CHOOSE (<i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i>)	Returns the <i>descriptive_col_expr</i> value of the matched row with the highest-sorted <i>quantifying_col_expr</i> value. For example, MAX_CHOOSE (product_price, product_name OF B) returns the product_name of the most expensive product in the rows that map to B. The <i>descriptive_col_expr</i> can have any data type. The <i>quantifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
MIN_CHOOSE (<i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i>)	Returns the <i>descriptive_col_expr</i> value of the matched row with the lowest-sorted <i>quantifying_col_expr</i> value. For example, MIN_CHOOSE (product_price, product_name OF B) returns the product_name of the least expensive product in the rows that map to B. The <i>descriptive_col_expr</i> can have any data type. The <i>quantifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
DUPCOUNT (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in the immediately preceding matched row. When <i>col_expr</i> is also the ORDER BY <i>col_expr</i> , this function returns the equivalent of ROW_NUMBER() - RANK().
DUPCOUNTCUM (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the cumulative duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in all preceding matched rows. When <i>col_expr</i> is also the ORDER BY <i>col_expr</i> , this function returns the equivalent of ROW_NUMBER() - DENSE_RANK().
ACCUMULATE ([DISTINCT CDISTINCT] <i>col_expr</i> OF <i>symbol_list</i> [DELIMITER ' <i>delimiter</i> '])	Returns, for each matched row, the concatenated values in <i>col_expr</i> , separated by <i>delimiter</i> . The default delimiter is ',' (a comma followed by a space). DISTINCT limits the concatenated values to distinct values. CDISTINCT limits the concatenated values to consecutive distinct values.

Function	Description
	<p>Note:</p> <p>The characters in <i>col_expr</i> must belong to the LATIN character set. If they are from the UNICODE character set, translate them to LATIN. For example:</p> <pre>TRANSLATE (col_expr USING UNICODE_TO_LATIN) as col_expr</pre>

You can compute an aggregate over more than one symbol. For example, `SUM (val OF ANY (A,B))` computes the sum of the values of the attribute *val* across all rows in the matched segment that map to A or B.

nPath Results Examples

nPath Results Example: FIRST, LAST_NOTNULL, MAX_CHOOSE, and MIN_CHOOSE

Input

trans1

userid	gender	ts	productname	productamt
1	M	2012-01-01 00:00:00	shoes	100
1	M	2012-02-01 00:00:00	books	300
1	M	2012-03-01 00:00:00	television	500
1	M	2012-04-01 00:00:00	envelopes	10
2		2012-01-01 00:00:00	bookcases	150
2		2012-02-01 00:00:00	tables	250
2	F	2012-03-01 00:00:00	appliances	1500
3	F	2012-01-01 00:00:00	chairs	400
3	F	2012-02-01 00:00:00	cellphones	600
3	F	2012-03-01 00:00:00	dvds	50

SQL Call

```
SELECT * FROM nPath (
  ON trans1 PARTITION BY userid ORDER BY ts
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A+')
```

```

Symbols (TRUE AS A)
Result (
  FIRST (userid OF A) AS Userid,
  LAST_NOTNULL (gender OF A) AS Gender,
  MAX_CHOOSE (productamt, productname OF A) AS Max_prod,
  MIN_CHOOSE (productamt, productname OF A) AS Min_prod
)
) ORDER BY 1;

```

Output

userid	gender	max_prod	min_prod
1	M	television	envelopes
2	F	appliances	bookcases
3	F	cellphones	dvds

nPath Results Example: FIRST and Three Forms of ACCUMULATE

Input

clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	null	home	06:59:13	Company1	100
1039	1	null	home	07:00:10	Company2	300
1039	1	television	checkout	07:00:12	Company2	500
1039	1	television	checkout	07:00:18	Company2	10
1039	1	envelopes	checkout	07:01:00	Company3	10
1039	1	null	checkout	07:01:10	Company3	10

SQL Call

```

SELECT * FROM nPath (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Symbols (
    pagetype='home' AS H,
    pagetype='checkout' AS C,
    pagetype <> 'home' AND pagetype <>'checkout' AS A

```

```

)
Pattern ('^H+.A*.C+$')
Result (
  FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
  FIRST (clicktime OF H) AS firsthome,
  FIRST (clicktime OF C) AS firstcheckout,
  ACCUMULATE (productname OF ANY (H,A,C) DELIMITER '*') AS products_accumulate,
  ACCUMULATE (CDISTINCT productname OF ANY (H,A,C) DELIMITER '$$') AS
cde_dup_products,
  ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS de_dup_products
)
) ORDER BY sessionid;

```

Output

sessionid	firsthome	firstcheckout	products_accumulate	cde_dup_products	de_dup_products
1	06:59:13	07:00:12	[null*null*television*television*envelopes*null]	[null\$\$television\$\$envelopes\$ \$null]	[null, television, envelopes]

nPath Results Example: FIRST, Three Forms of ACCUMULATE, COUNT, and NTH**Input**

The input table for this example is clicks, as in [nPath Results Example: FIRST and Three Forms of ACCUMULATE](#).

SQL Call

```
SELECT * FROM nPath (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Symbols (
    pagetype='home' AS H,
    pagetype='checkout' AS C,
    pagetype <> 'home' AND pagetype <>'checkout' AS A
  )
  Pattern ('^H+.A*.C+$')
  Result (
    FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
    FIRST (clicktime OF H) AS firsthome,
    FIRST (clicktime OF C) AS firstcheckout,
    ACCUMULATE (productname OF ANY (H,A,C)) AS products_accumulate,
    COUNT (DISTINCT productname OF ANY(H,A,C)) AS count_distinct_products,
    ACCUMULATE (CDISTINCT productname OF ANY (H,A,C)) AS
consecutive_distinct_products,
    ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS distinct_products,
    NTH (productname, -1 OF ANY(H,A,C)) AS nth
  )
) ORDER BY sessionid;
```

Output

sessionid	firsthome	firstcheckout	products_accumulate	count_distinct_products	consecutive_distinct_products	distinct_products	nth
1	06:59:13	07:00:12	[null, null, television, television, envelopes, null]	2	[null, television, envelopes, null]	[null, television, envelopes]	?

nPath Results Example: Combine Values from One Row with Values from the Next Row

Input

The input table is clickstream, as in [nPath Filters Example](#).

SQL Call

```
SELECT * FROM nPath (
  ON clickstream PARTITION BY userid ORDER BY userid, sessionid, clicktime
  USING
  Mode (OVERLAPPING)
  Pattern ('A.B')
  Symbols (TRUE AS A, TRUE AS B)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    FIRST (pagetype OF A) AS pageid,
    FIRST (pagetype OF B) AS next_pageid
  )
) ORDER BY sessionid;
```

Output

sessionid	pageid	next_pageid
1	home	view
1	view	view
1	checkout	view
1	checkout	checkout
1	view	checkout
1	view	view
2	checkout	view
2	home	view
2	view	view
2	view	checkout

nPath Differences on Aster Database and Teradata Database

Aster Database nPath	Teradata Database nPath
In a symbol, the Boolean expression TRUE, NOT TRUE, or <i>integer</i> can be enclosed in parentheses or quotation marks.	In a symbol, the Boolean expression TRUE, NOT TRUE, or <i>integer</i> cannot be enclosed in parentheses or quotation marks.
Aggregate functions compare strings using Unicode value of each character (lexicographic order), ignoring CHARACTER SET.	Aggregate functions compare strings using sort order, based on CHARACTER SET, CASESPECIFIC, and COLLATION.

AVG

Database	Syntax Element Data Type	Return Data Type
Aster	SMALLINT, INTEGER, or BIGINT	NUMERIC
	DOUBLE PRECISION, NUMERIC, or INTERVAL	Same as syntax element data type
Teradata	NUMERIC	DOUBLE PRECISION
	INTERVAL, or DATE without TIME or TIMESTAMP	Same as syntax element data type

COUNT

Database	Syntax Element Data Type	Return Data Type
Aster	Any	BIGINT
Teradata	Any	TD mode: INTEGER ANSI mode: Depends on MaxDecimal value in DBSControl—see following table.

MaxDecimal Value	Result Data Type	Result Data Type Format
0 or 15	NUMERIC(15,0)	-(15)9
18	NUMERIC(18,0)	-(18)9
38	NUMERIC(38,0)	-(38)9

MAX and MIN

Database	Syntax Element Data Type	Return Data Type
Aster	Any numeric, string, or DateTime type	Same as syntax element data type

Database	Syntax Element Data Type	Return Data Type
Teradata	Any numeric, character, DateTime or Interval data type, or BYTE	If not UDT: Same as syntax element data type UDT: Data type to which UDT is implicitly cast

SUM

Database	Syntax Element Data Type	Return Data Type
Aster	SMALLINT or INTEGER	BIGINT
	BIGINT	NUMERIC
	DOUBLE PRECISION	DOUBLE PRECISION
	NUMERIC or INTERVAL	Same as syntax element data type
Teradata	NUMERIC, INTERVAL, or DATE without TIME or TIMESTAMP	Same as syntax element data type, except for NUMERIC(<i>n,m</i>), which returns NUMERIC(<i>p,m</i>), where <i>p</i> depends on MaxDecimal value in DBSControl—see following table.
	CHARACTER or VARCHAR	DOUBLE PRECISION

MaxDecimal Value	<i>n</i>	<i>p</i>
0 or 15	$n \leq 15$	15
	$15 < n \leq 18$	18
	$n > 18$	38
18	$n \leq 18$	18
	$n > 18$	38
38	Any value	38

nPath Examples**Symbols and Symbol Predicates That Examples Use**

Symbol	Symbol Predicate
A	pageid IN (10, 25)
B	category = 10 OR (category = 20 AND pageid <> 33)
C	category IN (SELECT pageid FROM clicks1 GROUP BY userid HAVING COUNT(*) > 10)

Symbol	Symbol Predicate
D	referrer LIKE '%Amazon%'
X	TRUE

nPath ClickStream Data Examples

Input

This statement creates the input table of clickstream data that the examples use:

```
CREATE MULTISET TABLE clicks1 (
  userid INTEGER,
  sessionid INTEGER,
  pageid INTEGER,
  category INTEGER,
  ts TIMESTAMP FORMAT 'YYYY-MM-DDbHH:MI:SS',
  referrer VARCHAR (256),
  val FLOAT
) PRIMARY INDEX ( userid );
```

This statement gets the pageid for each row and the pageid for the next row in sequence:

```
SELECT dt.sessionid, dt.pageid, dt.next_pageid FROM nPath (
  ON clicks1 PARTITION BY sessionid ORDER BY ts
  USING
  Mode (OVERLAPPING)
  Pattern ('A.B')
  Symbols (TRUE AS A, TRUE AS B)
  Result (
    FIRST(sessionid OF A) AS sessionid,
    FIRST (pageid OF A) AS pageid,
    FIRST (pageid OF B) AS next_pageid
  )
) AS dt;
```

Example: Counting Preceding Rows in a Sequence

For each row, this invocation counts the number of preceding rows in a given sequence (including the current row). The ORDER BY clause specifies DESC because the pattern must be matched over the rows preceding the start row, while the semantics dictate that the pattern be matched over the rows following the start row.

```
SELECT dt.sessionid, dt.pageid, dt.countrank FROM nPath (
  ON clicks1 PARTITION BY sessionid ORDER BY ts DESC
  USING
```

```

Mode (OVERLAPPING)
Pattern ('A*')
Symbols (TRUE AS A)
Result (
  FIRST (sessionid OF A) AS sessionid,
  FIRST (pageid OF A) AS pageid,
  COUNT (*) OF A AS countrank
)
) AS dt;

```

Example: Complex Path Query

This query finds the user click-paths that start at pageid 50 and proceed either to pageid 80 or to pages in category 9 or category 10, finds the pageid of the last page in the path, counts the visits to page 80, and returns the maximum count for each last page, by which it sorts the output. The query ignores paths of fewer than five pages and pages for which category is less than zero.

```

SELECT dt.last_pageid, MAX(dt.count_page80) FROM nPath (
  ON (SELECT * FROM clicks1 WHERE category >= 0)
  PARTITION BY sessionid ORDER BY ts
  USING
  Pattern ('A.(B|C)*')
  Mode (OVERLAPPING)
  Symbols (
    pageid = 50 AS A,
    pageid = 80 AS B,
    pageid <> 80 AND category IN (9,10) AS C
  )
  Result (
    LAST(pageid OF ANY (A,B,C)) AS last_pageid,
    COUNT (*) OF B AS count_page80,
    COUNT (*) OF ANY (A,B,C) AS count_any
  )
) AS dt WHERE dt.count_any >= 5
GROUP BY dt.last_pageid
ORDER BY MAX(dt.count_page80);

```

nPath Range-Matching Examples

Whenever a user visits the home page and then visits checkout pages and buys increasingly expensive products, the nPath query returns the first purchase and the most expensive purchase.

nPath Example Input Table: aggregate_clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250
1040	2	Appliance	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16	Company3	400

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
				11:18:16		
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...

nPath Range-Matching Example: Accumulate Pages Visited in Each Session

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A*')
  Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF A) AS path
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home, home1, page1, checkout, checkout, home, home]
3	[home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

nPath Range-Matching Example: Find Sessions That Start at Home Page and Visit Page1**Input**

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('^H.A*.P1.A*')
  Symbols (pagetype='home' AS H, pagetype='page1' AS P1, TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF ANY(H,P1,A)) AS path
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home1, page1, checkout, checkout, home, home]

sessionid	path
3	[home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

nPath Range-Matching Example: Find Paths to Checkout Page for Purchases Over \$200

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A*.C+.A*')
  Symbols (
    productprice > 200 AND
    pagetype='checkout' AS C, TRUE AS A
  )
  Result (
    FIRST(sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF ANY(A,C)) AS path,
    AVG (productprice OF ANY(A,C)) AS totalsum
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	path	totalsum
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]	602.857142857143
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]	363.157894736842

nPath Range-Matching Example: Use OVERLAPPING Mode

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (OVERLAPPING)
  Pattern ('A.A')
  Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF A) AS path
  )
) AS dt ORDER BY dt.sessionid;
```

nPath Range-Matching Example Output

sessionid	path
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[checkout, home]
1	[page1, checkout]
1	[home1, page1]
1	[home, home1]
1	[home, home]
1	[home, home]
1	[page1, home]

sessionid	path
1	[home1, page1]
1	[home, home1]
1	[page1, home]
1	[home1, page1]
1	[home, home1]
2	[home, home]
2	[checkout, home]
2	[checkout, checkout]
...	...

nPath Range-Matching Example: Find First Product with Multiple Referrers in Any Session

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('REFERRER{2,}')
  Symbols (referrer IS NOT NULL AS REFERRER)
  Result (
    FIRST (sessionid OF REFERRER) AS sessionid,
    FIRST (productname OF REFERRER) AS product
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	product
1	envelopes
2	tables
3	bookcases

sessionid	product
4	tables
5	Appliances

nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.C{3,6}.D')
  Symbols (
    pagetype = 'home' AS H,
    pagetype='checkout' AS C,
    pagetype<>'home' AND pagetype<>'checkout' AS D
  )
  Result (
    FIRST (sessionid OF C) AS sessionid,
    max_choose (productprice, productname OF C) AS
    most_expensive_product,
    MAX (productprice OF C) AS max_price,
    min_choose (productprice, productname of C) AS
    least_expensive_product,
    MIN (productprice OF C) AS min_price)
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

nPath Range-Matching Example: Find Data for Sessions That Checked Out at Least 3 Products

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

Modify the previous query call in [nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products](#) to find sessions where the user checked out at least three products by changing the Pattern syntax element to:

```
Pattern ('H+.D*.C{3,}.D')
```

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.C{3,}.D')
  Symbols (
    pagetype = 'home' AS H,
    pagetype='checkout' AS C,
    pagetype<>'home' AND pagetype<>'checkout' AS D
  )
  Result (
    FIRST(sessionid OF C) AS sessionid,
    max_choose(productprice, productname OF C) AS
    most_expensive_product,
    MAX (productprice OF C) AS max_price,
    min_choose (productprice, productname OF C) AS
    least_expensive_product,
    MIN (productprice OF C) AS min_price
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

nPath Range-Matching Example: Multiple Partitioned Input Tables and Dimension Input Table

An e-commerce store wants to count the advertising impressions that lead to a user clicking an online advertisement. The example counts the online advertisements that the user viewed and the television advertisements that the user might have viewed.

Input**impressions**

userid	ts	imp
1	2012-01-01	ad1
1	2012-01-02	ad1
1	2012-01-03	ad1
1	2012-01-04	ad1
1	2012-01-05	ad1
1	2012-01-06	ad1
1	2012-01-07	ad1
2	2012-01-08	ad2
2	2012-01-09	ad2
2	2012-01-10	ad2
2	2012-01-11	ad2
...

clicks2

userid	ts	click
1	2012-01-01	ad1
2	2012-01-08	ad2
3	2012-01-16	ad3
4	2012-01-23	ad4
5	2012-02-01	ad5
6	2012-02-08	ad6
7	2012-02-14	ad7
8	2012-02-24	ad8
9	2012-03-02	ad9
10	2012-03-10	ad10
11	2012-03-18	ad11
12	2012-03-25	ad12

userid	ts	click
13	2012-03-30	ad13
14	2012-04-02	ad14
15	2012-04-06	ad15

tv_spots

ts	tv_imp
2012-01-01	ad2
2012-01-02	ad2
2012-01-03	ad3
2012-01-04	ad4
2012-01-05	ad5
2012-01-06	ad6
2012-01-07	ad7
2012-01-08	ad8
2012-01-09	ad9
2012-01-10	ad10
2012-01-11	ad11
2012-01-12	ad12
2012-01-13	ad13
2012-01-14	ad14
2012-01-15	ad15

SQL-MapReduce Call

The tables impressions and clicks have a user_id column, but the table tv_spots is only a record of television advertisements shown, which any user might have seen. Therefore, tv_spots must be a dimension table.

```
SELECT * FROM nPath (
  ON impressions PARTITION BY userid ORDER BY ts
  ON clicks2 PARTITION BY userid ORDER BY ts
  ON tv_spots DIMENSION ORDER BY ts
  USING
  Mode (NONOVERLAPPING)
```

```

Symbols (TRUE AS imp, TRUE AS click, TRUE AS tv_imp)
Pattern ('(imp|tv_imp)*.click')
Result (
  COUNT(* of imp) AS imp_cnt,
  COUNT (* of tv_imp) AS tv_imp_cnt
)
) AS dt ORDER BY dt.imp_cnt;

```

Output

dt.imp_cnt	tv_imp_cnt
18	
19	0
19	0
20	0
21	0
22	0
22	0
22	0
22	0
22	0
23	0
23	0
23	0
24	0
25	0

Pack

The Pack function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack complements the function [Unpack](#), but you can use it on any columns that meet the input requirements.

Note:

To use Pack and Unpack together, you must run both on NewSQL Engine platform. Pack and Unpack are incompatible with ML Engine functions Pack_MLE and Unpack_MLE.

Before packing columns, note their data types—you need them if you want to unpack the packed column.

Note:

- This function requires the UTF8 client character set.
- This function does not support locale-based formatting with the SDF file.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

Pack Syntax

```
SELECT * FROM Pack (
  ON { table | view | (query) }
  USING
  [ TargetColumns ('target_column' [,...]) ]
  [ Delimiter ('delimiter') ]
  [ IncludeColumnName ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]
  OutputColumn ('output_column')
) AS alias;
```

Pack Syntax Elements

TargetColumns

[Optional] Specify the names of the input table columns to pack into a single output column. Column names must be valid object names, which are defined in *Teradata Vantage™ SQL Fundamentals*, B035-1141.

These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

Delimiter

[Optional] Specify the delimiter—a single Unicode character in Normalization Form C (NFC)—that separates the virtual columns in the packed data. The *delimiter* is case-sensitive.

Default: ',' (comma)

IncludeColumnName

[Optional] Specify whether to label each virtual column value with its column name (making the virtual column *target_column:value*).

Default: 'true'

OutputColumn

Specify the name to give to the packed output column. The name must be a valid object name, as defined in *Teradata Vantage™ SQL Fundamentals*, B035-1141.

Pack Input

Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	[Column appears once for each specified <i>target_column</i> .] Column to pack, with other input columns, into single output column.
<i>other_input_column</i>	Any	[Column appears zero or more times.] Column to copy to output table. Typically, one such column contains row identifiers.

Pack Output

Output Table Schema

Column	Data Type	Description
<i>output_column</i>	VARCHAR Note: If a column of type DATE is packed into the output, its format is 'YYYY-MM-DD'.	Packed column.
<i>other_input_column</i>	Same as in input table	[Column appears once for each specified <i>other_input_column</i> .] Column copied from input table.

Pack Examples

Pack Examples Input

The input table, *ville_temperature* contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2010-01-01 00:00:00	35.1
2	Nashville	Tennessee	2010-01-01 01:00:00	36.2
3	Nashville	Tennessee	2010-01-01 02:00:00	34.5
4	Nashville	Tennessee	2010-01-01 03:00:00	33.6
5	Nashville	Tennessee	2010-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2010-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2010-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2010-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2010-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2010-01-01 07:00:00	32.4

Pack Example: Default Options

This example specifies the default options for Delimiter and IncludeColumnName.

Input

See [Pack Examples Input](#).

SQL Call

```
SELECT * FROM Pack (
  ON ville_temperature
  USING
  Delimiter(',')
  OutputColumn('packed_data')
  IncludeColumnName('true')
  TargetColumns('city', 'state', 'period', 'temp_f')
) AS dt ORDER BY 2;
```

Output

The columns specified by TargetColumns are packed in the column packed_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

packed_data	sn
city:Nashville,state:Tennessee,period:2010-01-01 00:00:00,temp_f:35.1	1
city:Nashville,state:Tennessee,period:2010-01-01 01:00:00,temp_f:36.2	2
city:Nashville,state:Tennessee,period:2010-01-01 02:00:00,temp_f:34.5	3
city:Nashville,state:Tennessee,period:2010-01-01 03:00:00,temp_f:33.6	4
city:Nashville,state:Tennessee,period:2010-01-01 04:00:00,temp_f:33.1	5
city:Knoxville,state:Tennessee,period:2010-01-01 03:00:00,temp_f:33.2	6
city:Knoxville,state:Tennessee,period:2010-01-01 04:00:00,temp_f:32.8	7
city:Knoxville,state:Tennessee,period:2010-01-01 05:00:00,temp_f:32.4	8
city:Knoxville,state:Tennessee,period:2010-01-01 06:00:00,temp_f:32.2	9
city:Knoxville,state:Tennessee,period:2010-01-01 07:00:00,temp_f:32.4	10

Pack Example: Nondefault Options

This example specifies the pipe character (|) for Delimiter and 'false' for IncludeColumnName.

Input

See [Pack Examples Input](#).

SQL Call

```
SELECT * FROM Pack (
  ON ville_temperature
  USING
  Delimiter ('|')
  OutputColumn ('packed_data')
  IncludeColumnName ('false')
  TargetColumns ('city', 'state', 'period', 'temp_f')
) AS dt ORDER BY 2;
```

Output

Virtual columns are separated by pipe characters and not labeled with their column names.

packed_data	sn
Nashville Tennessee 2010-01-01 00:00:00 35.1	1
Nashville Tennessee 2010-01-01 01:00:00 36.2	2

packed_data	sn
Nashville Tennessee 2010-01-01 02:00:00 34.5	3
Nashville Tennessee 2010-01-01 03:00:00 33.6	4
Nashville Tennessee 2010-01-01 04:00:00 33.1	5
Knoxville Tennessee 2010-01-01 03:00:00 33.2	6
Knoxville Tennessee 2010-01-01 04:00:00 32.8	7
Knoxville Tennessee 2010-01-01 05:00:00 32.4	8
Knoxville Tennessee 2010-01-01 06:00:00 32.2	9
Knoxville Tennessee 2010-01-01 07:00:00 32.4	10

Sessionize

The Sessionize function maps each click in a session to a unique session identifier. A *session* is a sequence of clicks by one user that are separated by at most *n* seconds.

The function is useful for both sessionization and detecting web crawler ("bot") activity. A typical use is to understand user browsing behavior on a web site.

Sessionize Syntax

```
SELECT * FROM Sessionize (
  ON { table | view | (query) }
  PARTITION BY expression [,...]
  ORDER BY order_column [,...]
  USING
  TimeColumn ('time_column')
  TimeOut (session_timeout)
  [ ClickLag (min_human_click_lag) ]
  [ EmitNull ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
) AS alias;
```

Sessionize Syntax Elements

TimeColumn

Specify the name of the input column that contains the click times.

Note:

The *time_column* must also be an *order_column*.

TimeOut

Specify the number of seconds at which the session times out. If session_timeout seconds elapse after a click, the next click starts a new session. The data type of session_timeout is DOUBLE PRECISION.

ClickLag

[Optional] Specify the minimum number of seconds between clicks for the session user to be considered human. If clicks are more frequent, indicating that the user is a bot, the function ignores the session. The *min_human_click_lag* must be less than *session_timeout*. The data type of *min_human_click_lag* is DOUBLE PRECISION.

Default behavior: The function ignores no session, regardless of click frequency.

EmitNull

[Optional] Specify whether to output rows that have NULL values in their session id and rapid fire columns, even if their *time_column* has a NULL value.

Default: 'false'

Sessionize Input

Input Table Schema

Column	Data Type	Description
<i>time_column</i>	TIME, TIMESTAMP, INTEGER, BIGINT, or SMALLINT	Click times (in milliseconds if data type is INTEGER, BIGINT, or SMALLINT).
<i>partition_column</i>	Any	[Column appears once for each specified <i>partition_column</i> .] Column by which input data is partitioned. Input data must be partitioned such that each partition contains all rows of an entity.
<i>order_column</i>	Any	[Column appears once for each specified <i>order_column</i> .] Column by which input data is ordered.

Note:

No input table column can have the name 'sessionid' or 'clicklag', because these are output table column names.

Tip:

To create a single timestamp column from separate date and time columns:

```
SELECT (datecolumn || ' ' || timecolumn)::timestamp AS mytimestamp FROM table;
```

Sessionize Output

Output Table Schema

Column	Data Type	Description
<i>input_column</i>	Same as in input table	Column copied from input table. Function copies every <i>input_column</i> to output table.
sessionid	INTEGER or BIGINT	Identifier that function assigned to session.
clicklag	BYTEINT	'1' if the session exceeded <i>min_human_click_lag</i> , '0' otherwise.

Sessionize Example

Input

sessionize_table

partition_id	clicktime	userid	productname	pagetype	referrer	productprice
1	1110000	333		Home	www.yahoo.com	
1	1112000	333	Ipod	Checkout	www.yahoo.com	200.2
1	1160000	333	Bose	Checkout		340
1	1200000	333		Home	www.google.com	
1	1203000	67403		Home	www.google.com	
1	1300000	67403		Home	www.google.com	
1	1301000	67403		Home		
1	1302000	67403		Home		
1	1340000	67403	Iphone	Checkout		650
1	1450000	67403	Bose	Checkout		750
1	1450200	80000		Home	godaddy.com	
1	1450600	80000	Bose	Checkout		340
1	1450800	80000	Itrip	Checkout		450
1	1452000	880000	Iphone	Checkout		650

SQL Call

```
SELECT * FROM Sessionize (
  ON sessionize_table PARTITION BY partition_id ORDER BY clicktime
```

```

USING
TimeColumn ('clicktime')
TimeOut (60)
ClickLag (0.2)
) ORDER BY partition_id, clicktime;

```

Output

partition_id	clicktime	userid	productname	pagetype	referrer	productprice	SESSIONID	CLICKLAG
1	1110000	333		Home	www.yahoo.com		0	f
1	1112000	333	Ipod	Checkout	www.yahoo.com	200.2	0	f
1	1160000	333	Bose	Checkout		340	0	f
1	1200000	333		Home	www.google.com		0	f
1	1203000	67403		Home	www.google.com		0	f
1	1300000	67403		Home	www.google.com		1	f
1	1301000	67403		Home			1	f
1	1302000	67403		Home			1	f
1	1340000	67403	Iphone	Checkout		650	1	f
1	1450000	67403	Bose	Checkout		750	2	f
1	1450200	80000		Home	godaddy.com		2	t
1	1450600	80000	Bose	Checkout		340	2	f
1	1450800	80000	Ittrip	Checkout		450	2	t
1	1452000	880000	Iphone	Checkout		650	2	f

StringSimilarity

The StringSimilarity function calculates the similarity between two strings, using the specified comparison method. The similarity is a value in the range [0, 1].

Note:

- This function requires the UTF8 client character set.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

- When comparing strings, the function assumes that they are in the same Unicode script in Normalization Form C (NFC).
- When used with this function, the ORDER BY clause supports only ASCII collation.

StringSimilarity Syntax

```
SELECT * FROM StringSimilarity (
  ON { table | view | (query) } [ PARTITION BY ANY ]
  USING
  ComparisonColumnPairs ('comparison_type (column1,column2[,constant])[ AS
output_column]')[,...]
  [ CaseSensitive ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}[,...]) ]
  [ Accumulate ('accumulate_column' [,...]) ]
) AS alias;
```

StringSimilarity Syntax Elements

ComparisonColumnPairs

Specify the names of the input table columns that contain strings to compare (*column1* and *column2*), how to compare them (*comparison_type*), and (optionally) a constant and the name of the output column for their similarity (*output_column*). The similarity is a value in the range [0, 1].

For *column1* and *column2*:

- If *column1* or *column2* includes any special characters (that is, characters other than letters, digits, or underscore (_)), surround the column name with double quotation marks. For example, if *column1* and *column2* are c(col1) and c(col2), respectively, specify them as "c(col1)" and "c(col2)".

If *column1* or *column2* includes double quotation marks, replace each double quotation mark with a pair of double quotation marks. For example, if *column1* and *column2* are c1"c and c2"c, respectively, specify them as "c1""c" and "c2""c".

Note:

These rules do not apply to *output_column*. For example, this is valid syntax:
ComparisonColumnPairs ('jaro ("c1""c", "c2""c") AS out"col1')

- If *column1* or *column2* supports more than 200 characters, you can cast it to VARCHAR(200), as in the following example; however, the string may be truncated. For information about the CAST operation, see *Teradata Vantage™ SQL Functions, Expressions, and Predicates*, B035-1145.

```
SELECT * FROM StringSimilarity (
  ON (
    SELECT id, CAST(a AS VARCHAR(200)) AS a, CAST(b AS VARCHAR(200)) AS b
    FROM max_varchar_strlen
  ) PARTITION BY ANY
  USING
  ComparisonColumnPairs ('ld(a,b) AS sim_fn')
```



```
Accumulate ('id')
) AS dt ORDER BY 1;
```

For *comparison_type*, use one of these values:

<i>comparison_type</i>	Description
'jaro'	Jaro distance.
'jaro_winkler'	Jaro-Winkler distance: 1 for an exact match, 0 otherwise. If you specify this comparison type, you can specify the value of factor p with <i>constant</i> . $0 \leq p \leq 0.25$. Default: $p = 0.1$
'n_gram'	N -gram similarity. If you specify this comparison type, you can specify the value of N with <i>constant</i> . Default: $N = 2$
'LD'	Levenshtein distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, or substitutions of individual characters.
'LDWS'	Levenshtein distance without substitution: Number of edits needed to transform one string into the other using only insertions or deletions of individual characters.
'OSA'	Optimal string alignment distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, substitutions, or transpositions of characters. A substring can be edited only once.
'DL'	Damerau-Levenshtein distance: Like 'OSA' except that a substring can be edited any number of times.
'hamming'	Hamming distance: For strings of equal length, number of positions where corresponding characters differ (that is, minimum number of substitutions needed to transform one string into the other). For strings of unequal length, -1.
'LCS'	Longest common substring: Length of longest substring common to both strings.
'jaccard'	Jaccard indexed-based comparison.
'cosine'	Cosine similarity.
'soundexcode'	Only for English strings: -1 if either string has a non-English character; otherwise, 1 if their soundex codes are the same and 0 otherwise.

Note:

The function ignores *constant* for every *comparison_type* except 'jaro_winkler' and 'n_gram'.

You can specify a different *comparison_type* for every pair of columns.

Default: *output_column* is 'sim_*i*', where *i* is the sequence number of the column pair.

CaseSensitive

[Optional] Specify whether string comparison is case-sensitive. You can specify either one value for all pairs or one value for each pair. If you specify one value for each pair, the *i*th value applies to the *i*th pair.

Default: 'false'

Accumulate

[Optional] Specify the names of input table columns to copy to the output table.

StringSimilarity Input

The table can have additional columns, but the function ignores them.

Column	Data Type	Description
<i>column1</i>	CHARACTER or VARCHAR	[Column appears once for each specified <i>column1</i> .] String to compare to string in <i>column2</i> .
<i>column2</i>	CHARACTER or VARCHAR	[Column appears once for each specified <i>column2</i> .] String to compare to string in <i>column1</i> .
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

Note:

If any *column1* or *column2* in the input table schema supports more than 200 characters, you must cast it to VARCHAR(200). See example in [StringSimilarity Syntax Elements](#).

StringSimilarity Output**Output Table Schema**

Column	Data Type	Description
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from input table.
<i>output_column</i>	DOUBLE PRECISION	[Column appears once for each column pair.] Similarity between strings in column pair.

StringSimilarity Example**Input**

strsimilarity_input

id	src_text1	src_text2	tar_text
1	astre	astter	aster

id	src_text1	src_text2	tar_text
2	hone	fone	phone
3	acqiese	acquire	acquiesce
4	AAAACCCCCGGGGA	CCCGGGAACCAACC	CCAGGGAAACCCAC
5	alice	allen	allies
6	angela	angle	angels
7	senter	center	centre
8	chef	cheap	chief
9	circus	circle	circuit
10	debt	debut	debris
11	deal	dell	lead
12	bare	bear	bear

SQL Call

```

SELECT * FROM StringSimilarity (
  ON strsimilarity_input PARTITION BY ANY
  USING
  ComparisonColumnPairs ('jaro (src_text1, tar_text) AS jaro1_sim',
                        'LD (src_text1, tar_text) AS ld1_sim',
                        'n_gram (src_text1, tar_text, 2) AS ngram1_sim',
                        'jaro_winkler (src_text1, tar_text, 0.1) AS jw1_sim'
  )
  CaseSensitive ('true')
  Accumulate ('id', 'src_text1', 'tar_text')
) AS dt ORDER BY id;

```

Output

Columns 1-3

id	src_text1	tar_text
1	astre	aster
2	hone	phone
3	acqiese	acquiesce
4	AAAACCCCCGGGGA	CCAGGGAAACCCAC
5	alice	allies

id	src_text1	tar_text
6	angela	angels
7	senter	centre
8	chef	chief
9	circus	circuit
10	debt	debris
11	deal	lead
12	bare	bear

Columns 4-7

jaro1_sim	ld1_sim	ngram1_sim	jw1_sim
0.9333333333333333	0.6	0.5	0.9533333333333333
0.9333333333333333	0.8	0.75	0.9333333333333333
0.925925925925926	0.777777777777778	0.5	0.948148148148148
0.824175824175824	0.214285714285714	0.384615384615385	0.824175824175824
0.822222222222222	0.5	0.4	0.857777777777778
0.888888888888889	0.833333333333333	0.8	0.933333333333333
0.822222222222222	0.5	0.4	0.822222222222222
0.933333333333333	0.8	0.5	0.946666666666667
0.849206349206349	0.714285714285714	0.666666666666667	0.90952380952381
0.75	0.5	0.4	0.825
0.666666666666667	0.5	0.333333333333333	0.666666666666667
0.833333333333333	0.5	0.333333333333333	0.85

SVMSParsePredict

The SVMSParsePredict function uses the model output by ML Engine SVMSParse function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

SVMSParsePredict Syntax

```

SELECT * FROM SVMSParsePredict (
  ON { table | view | (query) } AS InputTable PARTITION BY id_column
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('id_column')
  AttributeNameColumn ('attribute_name_column')
  [ AttributeValueColumn ('attribute_value_column') ]
  [ Accumulate ('accumulate_column'[,...]) ]
  [ OutputClassNum ('output_class_number') ]
) AS alias;

```

SVMSParsePredict Syntax Elements

IDColumn

Specify the name of the InputTable column that contains the identifiers of the test samples. The InputTable must be partitioned by this column.

AttributeNameColumn

Specify the name of the InputTable column that contains the attributes of the test samples.

AttributeValueColumn

[Optional] Specify the name of the InputTable column that contains the attribute values.

Default behavior: Each attribute has the value 1.

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

OutputClassNum

[Optional] Valid only for multiple-class models. Specify the number of class labels to appear in the output table, with its corresponding prediction confidence.

Default: 1

SVMSParsePredict Input

Table	Description
InputTable	Contains test data.
Model	Output by ML Engine SVMSParse function. Model is in binary format. To display its readable content, use ML Engine SVMSParseSummary function.

InputTable Schema

Column	Data Type	Description
<i>id_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), NUMERIC(p,a), VARCHAR, or VARCHAR(n)	Test sample identifier.
<i>attribute_name_column</i>	INTEGER, SMALLINT, BIGINT, VARCHAR, or VARCHAR(n)	Test sample attribute.
<i>attribute_value_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), or NUMERIC(p,a)	Attribute value.
<i>accumulate_column</i>		[Column appears once for each specified <i>accumulate_column</i> .] Column to copy to output table.

Model Table Schema

Column	Data Type	Description
classid	INTEGER or BIGINT	Identifier of class of model attribute.
weights	VARBYTE	Weight of model attribute.

SVMSParsePredict Output**Output Table Schema**

The table has the predicted class of each test sample.

Column	Data Type	Description
<i>id_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), NUMERIC(p,a), VARCHAR, or VARCHAR(n)	Test sample identifier.
predict_value	VARCHAR	Predicted class of test sample.
predict_confidence	DOUBLE PRECISION	Prediction confidence, a value between 0 and 1, computed by formula that follows this table. The higher the value, the more dependable the prediction.
<i>accumulate_column</i>	Any	[Column appears once for each specified <i>accumulate_column</i> .] Column copied from InputTable.

Formula for predict_confidence

$$confidence = \frac{I}{1 + \exp(-\sum w_i x_i)}$$

where i is the attribute id, x_i is the value of attributes in the sample, and w_i is the weight of attribute i .

SVMSParsePredict Example**Input**

- InputTable: svm_iris_input_test
- Model: svm_iris_model, output by ML Engine SVMSParse function

The model is in binary format. To display its readable content, use ML Engine SVMSParseSummary function.

svm_iris_input_test

id	species	attribute	value
5	setosa	sepal_length	5.0
5	setosa	sepal_width	3.6
5	setosa	petal_length	1.4
5	setosa	petal_width	0.2
10	setosa	sepal_length	4.9
10	setosa	sepal_width	3.1
10	setosa	petal_length	1.5
10	setosa	petal_width	0.1
15	setosa	sepal_length	5.8
15	setosa	sepal_width	4.0
15	setosa	petal_length	1.2
15	setosa	petal_width	0.2
...

SQL Call

```
CREATE MULTISET TABLE svm_iris_predict_out AS (
  SELECT * FROM SVMSParsePredict (
    ON svm_iris_input_test AS InputTable PARTITION BY id
```

```

ON svm_iris_model AS Model DIMENSION
USING
  IDColumn ('id')
  AttributeNameColumn ('attribute')
  AttributeValueColumn ('value')
  Accumulate ('species')
) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM svm_iris_predict_out ORDER BY id;
```

id	predict_value	predict_confidence	species
5	setosa	0.878736053291771	setosa
10	setosa	0.827684323576856	setosa
15	setosa	0.933727152238982	setosa
...

Prediction Accuracy

This query returns the prediction accuracy:

```

SELECT (SELECT count(id)
FROM svm_iris_predict_out
WHERE predict_value = species)/(
  SELECT count(id) FROM svm_iris_predict_out) AS prediction_accuracy;

```

prediction_accuracy
0.946666666666667

Unpack

The Unpack function unpacks data from a single packed column into multiple columns. The packed column is composed of multiple virtual columns, which become the output columns. To determine the virtual columns, the function must have either the delimiter that separates them in the packed column or their lengths.

Unpack complements the function [Pack](#), but you can use it on any packed column that meets the input requirements.

Note:

- To use Pack and Unpack together, you must run both on NewSQL Engine platform. Pack and Unpack are incompatible with ML Engine functions Pack_MLE and Unpack_MLE.
- This function requires the UTF8 client character set.
- This function does not support locale-based parsing with the SDF file.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ NewSQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Unpack Syntax

```
SELECT * FROM Unpack (
  ON { table | view | (query) }
  USING
  TargetColumn ('target_column')
  OutputColumns ('output_column' [...])
  OutputDataTypes ('datatype' [...])
  [ Delimiter ('delimiter') ]
  [ ColumnLength ('column_length' [...]) ]
  [ Regex ('regular_expression') ]
  [ RegexSet ('group_number') ]
  [ IgnoreInvalid ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
) AS alias;
```

Unpack Syntax Elements

TargetColumn

Specify the name of the input column that contains the packed data.

OutputColumns

Specify the names to give to the output columns, in the order in which the corresponding virtual columns appear in *target_column*. The names must be valid object names, as defined in *Teradata Vantage™ SQL Fundamentals*, B035-1141.

Note:

If you specify fewer output column names than there are virtual input columns, the function ignores the extra virtual input columns. That is, if the packed data contains $x+y$ virtual columns and the `OutputColumns` syntax element specifies x output column names, the function assigns the names to the first x virtual columns and ignores the remaining y virtual columns.

OutputDataTypes

Specify the datatypes of the unpacked output columns. Supported data types are VARCHAR, INTEGER, DOUBLE PRECISION, TIME, DATE, and TIMESTAMP.

If `OutputDataTypes` specifies only one value and `OutputColumns` specifies multiple columns, the specified value applies to every *output_column*.

If `OutputDataTypes` specifies multiple values, it must specify a value for each *output_column*. The *nth datatype* corresponds to the *nth output_column*.

Note:

The function can output only 16 VARCHAR columns.

Delimiter

[Optional] Specify the delimiter—a single Unicode character in Normalization Form C (NFC)—that separates the virtual columns in the packed data. The *delimiter* is case-sensitive.

Note:

Do not specify both this syntax element and the `ColumnLength` syntax element. If the virtual columns are separated by a delimiter, specify the delimiter with this syntax element; otherwise, specify the `ColumnLength` syntax element.

Default: ',' (comma)

ColumnLength

[Optional] Specify the lengths of the virtual columns; therefore, to use this syntax element, you must know the length of each virtual column.

If `ColumnLength` specifies only one value and `OutputColumns` specifies multiple columns, the specified value applies to every *output_column*.

If `ColumnLength` specifies multiple values, it must specify a value for each *output_column*. The *nth datatype* corresponds to the *nth output_column*. However, the last *output_column* can be an asterisk (*), which represents a single virtual column that contains the remaining data. For example, if the first three virtual columns have the lengths 2, 1, and 3, and all remaining data belongs to the fourth virtual column, you can specify `ColumnLength ('2', '1', '3', *)`.

Note:

Do not specify both this syntax element and the Delimiter syntax element.

Regex

[Optional] Specify a regular expression that describes a row of packed data, enabling the function to find the data values.

A row of packed data contains a data value for each virtual column, but the row might also contain other information (such as the virtual column name). In the *regular_expression*, each data value is enclosed in parentheses.

For example, suppose that the packed data has two virtual columns, age and sex, and that one row of packed data is age:34, sex:ma1e. The *regular_expression* that describes the row is `'.*: (.*)'`. The `'.*:'` matches the virtual column names, age and sex, and the `'(.*)'` matches the values, 34 and male.

To represent multiple data groups in *regular_expression*, use multiple pairs of parentheses. Without parentheses, the last data group in *regular_expression* represents the data value (other data groups are assumed to be virtual column names or unwanted data). If a different data group represents the data value, specify its group number with the RegexSet syntax element.

Default: `'(.*)'`, which matches the whole string (between delimiters, if any). When applied to the preceding sample row, the default *regular_expression* causes the function to return 'age:34' and 'sex:ma1e' as data values.

RegexSet

[Optional] Specify the ordinal number of the data group in *regular_expression* that represents the data value in a virtual column.

Default behavior: The last data group in *regular_expression* represents the data value. For example, suppose that *regular_expression* is `'([a-zA-Z]*):(.*)'`. If *group_number* is '1', `'([a-zA-Z]*)'` represents the data value. If *group_number* is '2', `'(.*)'` represents the data value.

Maximum: 30

IgnoreInvalid

[Optional] Specify whether the function ignores rows that contain invalid data.

Default: 'false' (The function fails if it encounters a row with invalid data.)

Unpack Input

Input Table Schema

Column	Data Type	Description
<i>target_column</i>	CHARACTER, VARCHAR, or CLOB	Packed data.

Unpack Output

Output Table Schema

Column	Data Type	Description
<i>output_column</i>	Specified by OutputDataTypes syntax element.	[Column appears once for each specified <i>output_column</i> .] Unpacked column.
<i>other_input_column</i>	Same as in input table	[Column appears once for each input table column not specified by TargetColumn syntax element.] Column copied from input table. This column (or these columns) follow output columns.

Unpack Examples

Unpack Example: Delimiter Separates Virtual Columns

Input

The input table, `ville_tempdata`, is a collection of temperature readings for two cities, Nashville and Knoxville, in the state of Tennessee. In the column of packed data, the delimiter comma (,) separates the virtual columns. The last row contains invalid data.

`ville_tempdata`

sn	packed_temp_data
10	Nashville,Tennessee,35.1
11	Nashville,Tennessee,36.2
12	Nashville,Tennessee,34.5
13	Nashville,Tennessee,33.6
14	Nashville,Tennessee,33.1
15	Nashville,Tennessee,33.2
16	Nashville,Tennessee,32.8
17	Nashville,Tennessee,32.4
18	Nashville,Tennessee,32.2
19	Nashville,Tennessee,32.4
20	Thisisbaddata

SQL Call

```

SELECT * FROM Unpack (
  ON ville_tempdata
  USING
  TargetColumn ('packed_temp_data')
  OutputColumns ('city', 'state', 'temp_f')
  OutputDataTypes ('varchar', 'varchar', 'real')
  Delimiter (',')
  Regex ('(.*)')
  RegexSet (1)
  IgnoreInvalid ('true')
) AS dt ORDER BY sn;

```

Note:

Because comma is the default delimiter, the Delimiter syntax element in the preceding call is optional.

Output

Because of IgnoreInvalid ('true'), the function did not fail when it encountered the row with invalid data, but it did not output that row.

city	state	temp_f	sn
Nashville	Tennessee	3.510000000000000E 001	10
Nashville	Tennessee	3.620000000000000E 001	11
Nashville	Tennessee	3.450000000000000E 001	12
Nashville	Tennessee	3.360000000000000E 001	13
Nashville	Tennessee	3.310000000000000E 001	14
Nashville	Tennessee	3.320000000000000E 001	15
Nashville	Tennessee	3.280000000000000E 001	16
Nashville	Tennessee	3.240000000000000E 001	17
Nashville	Tennessee	3.220000000000000E 001	18
Nashville	Tennessee	3.240000000000000E 001	19

Unpack Example: No Delimiter Separates Virtual Columns

Input

The input table, ville_tempdata1, is like the input table for the previous example, except that no delimiter separates the virtual columns in the packed data. To enable the function to determine the virtual columns, the function call specifies the column lengths.

ville_tempdata1

sn	packed_temp_data
10	NashvilleTennessee35.1
11	NashvilleTennessee36.2
12	NashvilleTennessee34.5
13	NashvilleTennessee33.6
14	NashvilleTennessee33.1
15	NashvilleTennessee33.2
16	NashvilleTennessee32.8
17	NashvilleTennessee32.4
18	NashvilleTennessee32.2
19	NashvilleTennessee32.4
20	Thisisbaddata

SQL Call

```
SELECT * FROM Unpack (
  ON ville_tempdata1
  USING
    TargetColumn ('packed_temp_data')
    OutputColumns ('city', 'state', 'temp_f')
    OutputDataTypes ('varchar', 'varchar', 'real')
    ColumnLength ('9', '9', '4')
    Regex ('(.*)')
    RegexSet (1)
    IgnoreInvalid ('true')
) AS dt ORDER BY sn;
```

Output

city	state	temp_f	sn
Nashville	Tennessee	3.51000000000000E 001	10

city	state	temp_f	sn
Nashville	Tennessee	3.62000000000000E 001	11
Nashville	Tennessee	3.45000000000000E 001	12
Nashville	Tennessee	3.36000000000000E 001	13
Nashville	Tennessee	3.31000000000000E 001	14
Nashville	Tennessee	3.32000000000000E 001	15
Nashville	Tennessee	3.28000000000000E 001	16
Nashville	Tennessee	3.24000000000000E 001	17
Nashville	Tennessee	3.22000000000000E 001	18
Nashville	Tennessee	3.24000000000000E 001	19

Unpack Example: More Input Columns than Output Columns

Input

The input table is ville_tempdata1, as in [Unpack Example: No Delimiter Separates Virtual Columns](#). Its packed_temp_data column has three virtual columns.

SQL Call

The OutputColumns syntax element specifies only two output column names.

```
SELECT * FROM Unpack (
  ON ville_tempdata1
  USING
    TargetColumn ('packed_temp_data')
    OutputColumns ('city', 'state')
    OutputDataTypes ('varchar', 'varchar')
    ColumnLength ('9', '9')
    Regex ('(.*)')
    RegexSet (1)
    IgnoreInvalid ('true')
) AS dt ORDER BY sn;
```

Output

The output table has columns for the first two virtual input columns, but not for the third.

city	state	sn
Nashville	Tennessee	10

city	state	sn
Nashville	Tennessee	11
Nashville	Tennessee	12
Nashville	Tennessee	13
Nashville	Tennessee	14
Nashville	Tennessee	15
Nashville	Tennessee	16
Nashville	Tennessee	17
Nashville	Tennessee	18
Nashville	Tennessee	19

Additional Information

Changes and Additions

Date	Release	Description
July 2019	16.20 Feature Update 2	Added these functions: <ul style="list-style-type: none"> • Antiselect • MovingAverage • NGramSplitter • Pack • StringSimilarity • Unpack
May 2018	16.20 Feature Update 1	Added these functions: <ul style="list-style-type: none"> • Attribution • DecisionTreePredict • GLMPredict • NaiveBayesPredict • NaiveBayesTextClassifierPredict • RandomForestPredict • SVMSParsePredict

Teradata Links

Link	Description
https://docs.teradata.com/	Teradata documentation (HTML)
https://www.info.teradata.com	Teradata documentation (PDF)
https://access.teradata.com	One-stop source for Teradata community support, software downloads, and product information. Log in for customer access to: <ul style="list-style-type: none"> • Community support • Software updates • Knowledge articles • Orange Books
https://www.teradata.com/products-and-services/TEN	Teradata education network
https://community.teradata.com	Link to Teradata community (also available from the customer portal)

Related Documentation

Title	Publication ID
<i>Teradata Vantage™ NewSQL Engine Release Summary</i>	B035-1098
<i>Teradata Vantage™ SQL Fundamentals</i>	B035-1141
<i>Teradata Vantage™ SQL Functions, Expressions, and Predicates</i>	B035-1145
<i>Teradata Vantage™ SQL Data Manipulation Language</i>	B035-1146
<i>Teradata Vantage™ NewSQL Engine International Character Set Support</i>	B035-1125
<i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i>	B700-4003
<i>Teradata Aster® Database User Guide</i>	