



Aster Analytics Foundation

User Guide

Release Number 5.10

Product ID: B700-1000-510K

July 2013

The product or products described in this book are licensed products of Teradata Corporation or its affiliates.

Teradata, Active Data Warehousing, Active Enterprise Intelligence, Applications-Within, Aprimo, Aprimo Marketing Studio, Aster, BYNET, Claraview, DecisionCast, Gridscale, MyCommerce, Raising Intelligence, Smarter. Faster. Wins., SQL-MapReduce, Teradata Decision Experts, "Teradata Labs" logo, "Teradata Raising Intelligence" logo, Teradata ServiceConnect, Teradata Source Experts, "Teradata The Best Decision Possible" logo, The Best Decision Possible, WebAnalyst, and Xkoto are trademarks or registered trademarks of Teradata Corporation or its affiliates in the United States and other countries.

Adaptec and SCSISelect are trademarks or registered trademarks of Adaptec, Inc.

AMD Opteron and Opteron are trademarks of Advanced Micro Devices, Inc.

Apache, Apache Hadoop, Hadoop, and the yellow elephant logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

Apple, Mac, and OS X all are registered trademarks of Apple Inc.

Axeda is a registered trademark of Axeda Corporation. Axeda Agents, Axeda Applications, Axeda Policy Manager, Axeda Enterprise, Axeda Access, Axeda Software Management, Axeda Service, Axeda ServiceLink, and Firewall-Friendly are trademarks and Maximum Results and Maximum Support are servicemarks of Axeda Corporation.

Data Domain, EMC, PowerPath, SRDF, and Symmetrix are registered trademarks of EMC Corporation.

GoldenGate is a trademark of Oracle.

Hewlett-Packard and HP are registered trademarks of Hewlett-Packard Company.

Hortonworks, the Hortonworks logo and other Hortonworks trademarks are trademarks of Hortonworks Inc. in the United States and other countries.

Intel, Pentium, and XEON are registered trademarks of Intel Corporation.

IBM, CICS, RACF, Tivoli, and z/OS are registered trademarks of International Business Machines Corporation.

Linux is a registered trademark of Linus Torvalds.

LSI is a registered trademark of LSI Corporation.

Microsoft, Active Directory, Windows, Windows NT, and Windows Server are registered trademarks of Microsoft Corporation in the United States and other countries.

NetVault is a trademark or registered trademark of Quest Software, Inc. in the United States and/or other countries.

Novell and SUSE are registered trademarks of Novell, Inc., in the United States and other countries.

Oracle, Java, and Solaris are registered trademarks of Oracle and/or its affiliates.

QLogic and SANbox are trademarks or registered trademarks of QLogic Corporation.

Quantum and the Quantum logo are trademarks of Quantum Corporation, registered in the U.S.A. and other countries.

Red Hat is a trademark of Red Hat, Inc., registered in the U.S. and other countries. Used under license.

SAS and SAS/C are trademarks or registered trademarks of SAS Institute Inc.

SPARC is a registered trademark of SPARC International, Inc.

Symantec, NetBackup, and VERITAS are trademarks or registered trademarks of Symantec Corporation or its affiliates in the United States and other countries.

Unicode is a registered trademark of Unicode, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

THE INFORMATION CONTAINED IN THIS DOCUMENT 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 NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL TERADATA CORPORATION BE LIABLE FOR ANY INDIRECT, DIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST SAVINGS, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The information contained in this document 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.

Information contained in this document may contain technical inaccuracies or typographical errors. Information may be changed or updated without notice. Teradata Corporation may also make improvements or changes in the products or services described in this information at any time without notice.

To maintain the quality of our products and services, we would like your comments on the accuracy, clarity, organization, and value of this document. Please email: teradata-books@lists.teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed non-confidential. Teradata Corporation will have no obligation of any kind with respect to Feedback and will be free to use, reproduce, disclose, exhibit, display, transform, create derivative works of, and distribute the Feedback and derivative works thereof without limitation on a royalty-free basis. Further, Teradata Corporation will be free to use any ideas, concepts, know-how, or techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, or marketing products or services incorporating Feedback.

Copyright © 2000-2013 by Teradata Corporation. All Rights Reserved.

Preface

This guide provides instructions for users and administrators of Aster Analytics 5.10. If you're using a different version, you must download a different edition of this guide.

The following additional resources are available:

- Additional Aster Database documentation:
- Aster Database upgrades, clients and other packages:
<http://downloads.teradata.com/download/tools>
- Documentation for existing customers with a Teradata @ Your Service login:
<http://tays.teradata.com/>
- Documentation that is available to the public:
<http://www.info.teradata.com/>

Conventions Used in This Guide

This document assumes that the reader is comfortable working in Windows and Linux/UNIX environments. Many sections assume you are familiar with SQL.

This document uses the following typographical conventions.

Typefaces

Command line input and output, commands, program code, filenames, directory names, and system variables are shown in a monospaced font. Words in *italics* indicate an example or placeholder value that you must replace with a real value. **Bold type** is intended to draw your attention to important or changed items. Menu navigation and user interface elements are shown using the **User Interface Command** font.

SQL Text Conventions

In the SQL synopsis sections, we follow these conventions:

- Square brackets ([and]) indicate one or more optional items.
- Curly braces ({ and }) indicate that you must choose an item from the list inside the braces. Choices are separated by vertical lines (|).

- An ellipsis (...) means the preceding element can be repeated.
- A comma and an ellipsis (, ...) means the preceding element can be repeated in a comma-separated list.
- In command line instructions, SQL commands and shell commands are typically written with no preceding prompt, but where needed the default Aster Database SQL prompt is shown: beehive=>

Command Shell Text Conventions

For shell commands, the prompt is usually shown. The \$ sign introduces a command that's being run by a non-root user:

```
$ ls
```

The # sign introduces a command that's being run as root:

```
# ls
```

Contact Teradata Global Technical Support (GTS)

For assistance and updated documentation, contact Teradata Global Technical Support (GTS):

- **Support Portal:** <http://tays.teradata.com/>
- **International:** 212-444-0443
- **US Customers:** 877-698-3282
- **Toll Free Number:** 877-MyT-Data

About Teradata Aster

Teradata Aster provides data management and advanced analytics for diverse and big data, enabling the powerful combination of cost-effective storage and ultra-fast analysis of relational and non-relational data. Teradata Aster is a division of Teradata and is headquartered in San Carlos, California.

For more information, go to <http://www.asterdata.com>

About This Document

This is the “Aster Analytics Foundation User Guide,” version 5.10, edition 1. This edition covers Aster Analytics version 5.10 and was published August 6, 2013 11:45 am.

Get the latest edition of this guide! The *Aster Analytics Foundation User Guide* is updated frequently. You can find the latest edition at <http://tays.teradata.com/>

Document revision history:

June 2009: 3.0.3; November 30, 2009: 4.0; January 4, 2010: 4.0.1; February 9, 2010: 4.0.2
May 31, 2010: 4.5; September 22, 2010: 4.5.1; October 18, 2010: 4.5.1-hp2
April, 2011: 4.6; May, 2011: 4.6-hp1; June, 2011: 4.6-hp2; July, 2011: 4.6-hp3
September, 2011: 4.6.1;
November, 2011: 4.6.3; September, 2012: 4.6.3HP0A; November 2012: 4.6.3HP1
July, 2012: 5.0; October, 2012: 5.0-Release 1; January 2013: 5.0-Release 2
July 2013: 5.10

Table of Contents

Preface	5
Conventions Used in This Guide	5
Typefaces	5
SQL Text Conventions	5
Command Shell Text Conventions	6
Contact Teradata Global Technical Support (GTS)	6
About Teradata Aster	6
About This Document	6
Chapter 1: Introduction	24
Analytics at Scale: Full Dataset Analysis	24
Introduction to SQL-MapReduce	25
What is MapReduce?	25
Aster Database SQL MapReduce	26
SQL-MapReduce Query Syntax	27
SQL-MR with Multiple Inputs	29
Benefits of Multiple Inputs	29
How Multiple Inputs are Processed	30
Types of SQL-MR Inputs	30
Semantic Requirements for SQL-MR Functions	31
Use Cases and Examples for Multiple Inputs	32
SQL-MR Multiple Input FAQ	38
List of Aster Analytical Functions by Category	40
Aster Analytical Function Descriptions	43
Time Series, Path, and Attribution Analysis	43
Pattern Matching with Teradata Aster nPath	44
Statistical Analysis	44
Text Analysis	45
Cluster Analysis	46
Naive Bayes	46
Decision Trees	47
Association Analysis	47
Graph Analysis	47

Data Transformation	48
Visualization Functions	48
Aster Database Utilities.....	49

Chapter 2: Installing Analytical Functions..... 50

Aster Analytical Function Versioning	50
Installation Procedure	51
Download the Analytics Foundation Bundle.....	52
Download Beta Analytic Functions.....	52
Install the Functions	52
Upgrade the Functions.....	53
Set Permissions to Allow Users to Run Functions.....	53
Test the Functions.....	54
Using the \install Command in ACT	54
Installing a function in a specific schema.....	56
Installing Aster Database Driver-Based Analytical Functions	56
Required Components	56
Installing the SQL-MapReduce Functions in Aster Database.....	57
Installing the JDK on the Client System	57
Installing the Aster SQL-MapReduce API on the Client System	57
Testing the Installation of a JDBC-Based Analytics Function	58
Using Double-Quotes in Database Object Names	58
Creating a Timestamp Column	59
Best Practices	59
Error Message Delays.....	59

Chapter 3: Time Series, Path, and Attribution Analysis 60

Cumulative Moving Average	61
Summary	61
Background	61
Usage	61
Input Data	62
Output	62
Example	62
Error Messages	64
SAX	64
Summary	64

Background	64
Usage	66
Input.....	68
Output	69
Example	69
Path Generator.....	72
Summary	72
Background	72
Usage	73
Input Data	73
Output	73
Example	74
Error Messages	75
Path Starter.....	75
Summary	75
Background	75
Usage	75
Example	76
Path Summarizer.....	78
Summary	78
Background	78
Usage	78
Example	80
Error Messages	81
Sessionization.....	82
Background	82
Usage	82
Example	83
Error Messages	84
Attribution	85
Summary	85
Permissions	85
Attribution (Multiple Input)	85
Attribution (Single Input)	92
FrequentPaths.....	104
Summary	104
Background	104
Usage	108
Examples	111
Using Teradata Aster nPath and FrequentPaths to Select Sequences	121
Error Messages	123
DWT	125

Summary	125
Background	125
Usage	126
Examples	130
Error Messages	134
IDWT	135
Summary	135
Background	135
Usage	136
Examples	139
Error Messages	140
DWT2d	141
Summary	141
Background	141
Usage	142
Example	147
Error Messages	148
IDWT2d	150
Summary	150
Background	150
Usage	150
Example	152
Error Messages	154

Chapter 4: Pattern Matching with Teradata Aster nPath ... 156

What is Teradata Aster nPath?	156
Applications of Teradata Aster nPath	157
Multiple Input Support	158
nPath	158
Permissions	158
Syntax (version 1.0)	158
Patterns, Symbols, and Operators in Teradata Aster nPath	162
Patterns	162
Symbols	166
Operators	169
Nesting parentheses	170
Anchors	170
Results: Applying an SQL Aggregate	170
Teradata Aster nPath Examples	173
Clickstream Data	173

Lead	174
Rank	175
Complex Path Query	175
Range-Matching Examples.	176
Multiple Inputs with Partitioned Inputs.	178
Multiple Inputs with Dimensional Input.....	180
Teradata Aster nPath Visualization Examples	181
Chapter 5: Statistical Analysis.....	182
Approximate Distinct Count (count_approx_distinct).....	183
Summary	183
Background	183
Usage	183
Output	184
Example	185
Error Messages	185
Approximate Percentile (approx percentile).....	186
Summary	186
Background	186
Usage	186
Output	187
Example	188
Example Output from Approximate Percentile.....	189
Correlation (stats correlation)	189
Summary	189
Usage	189
Example	190
Example Output from Correlation Reduce	191
Error Messages	191
Histogram.....	191
Summary	191
Usage	191
Output	193
Example	193
Algorithm.....	194
Error Messages	195
Enhanced Histogram Function.....	195
Summary	195
hist_map	196
Usage	196

hist_reduce	197
Usage	197
Example	197
Linear Regression (stats linear reg)	203
Summary	203
Usage	203
Example	203
Notes	204
Error Messages	204
Logistic Regression (deprecated)	205
Logistic Predict (deprecated)	205
Generalized Linear Model (stats glm)	205
Summary	205
Background	206
Usage	207
Arguments	208
Input	210
Onscreen Output	210
Output Table	212
Examples	213
Error Messages	219
Generalized Linear Model Prediction (glmpredict)	220
Summary	220
Usage	220
Arguments	220
Output	221
Example	221
Principal Component Analysis (PCA)	223
Summary	223
Background	223
Usage	224
Input Data	225
Output	225
Example	225
Simple Moving Average (stats smavg)	227
Summary	227
Background	227
Usage	227
Example	228
Error Messages	230
Weighted Moving Average (stats wmavg)	230
Summary	230

Background	230
Usage	231
Example	232
Error Messages	233
Exponential Moving Average (stats emavg)	234
Summary	234
Background	234
Usage	234
Example	235
Error Messages	237
Volume-Weighted Average Price (stats vwap)	237
Summary	237
Background	238
Usage	238
Example Query 1.....	239
Example Query 2.....	240
Example 2 Output from VWAP	240
Error Messages	241
K-Nearest Neighbor Classification Algorithm (knn)	241
Summary	241
Background	242
Usage	242
Example	244
Error Messages	246
Support Vector Machines	246
Summary	246
Background	246
Implementation Notes	247
svm	247
Usage	248
Example	248
svm_predict	249
Usage	249
Examples	250
ConfusionMatrix	256
Summary	256
Usage	256
Percentile	259
Summary	259
Background	259
Usage	259
Example	260

Distribution Matching.....	262
distnmatch (“Hypothesis Test” Mode)	262
Summary	262
Usage	263
Examples	267
distnmatch (“Best Match” Mode)	277
Summary	277
Background	277
Usage	278
Examples	282
LARS Functions	286
SUMMARY	286
Background	286
Implementation Notes	287
lars.....	287
SUMMARY	287
Usage	287
Examples	289
larspredict.....	296
Summary	296
Usage	296
Example	297
Sample.....	299
Summary	299
Usage	299
FMeasure.....	305
Summary	305
Background	305
Usage	305
Arguments	305
Example	306
<hr/>	
Chapter 6: Text Analysis	308
Levenshtein Distance	308
Summary	308
Background	308
Usage	308
Example	309
nGram.....	310
Summary	310

Background	310
Usage	311
Example	312
Error Messages	313
Text Classifier.....	314
Summary	314
Usage	314
TextClassifierTrainer	314
Summary	314
Background	314
Usage	315
Example for TextClassifierTrainer	318
TextClassifier.....	319
Example for TextClassifier	320
TextClassifierEvaluator.....	321
Example for TextClassifierEvaluator	322
Text Parser (text_parser).....	323
Summary	323
Background	323
Usage	323
Example	326
Error Messages	328
Named Entity Recognition (NER)	328
Summary	328
Background	329
Usage	329
FindNamedEntity	329
Example for FindNamedEntity	333
TrainNamedEntityFinder.....	334
Example for TrainNamedEntityFinder	335
EvaluateNamedEntityFinderRow and EvaluateNamedEntityFinderPartition	336
Example for EvaluateNamedEntityFinder	337
Sentiment Extraction Functions.....	338
Summary	338
Usage	339
ExtractSentiment.....	339
Example for ExtractSentiment	342
EvaluateSentimentExtractor	344
Example for EvaluateSentimentExtractor	346
TrainSentimentExtractor	346
Example for TrainSentimentExtractor	348
Naive Bayes Text Classifier	349

Summary	349
Background	349
Naive Bayes Text (naiveBayesText).....	349
Arguments	350
Naive Bayes Text Predict (naiveBayesTextPredict)	351
Arguments	351
Example	351
TextTokenizer	361
Summary	361
Background	361
Arguments	362
Example	362

Chapter 7: Cluster Analysis.....	364
kmeans	364
Summary	364
Background	365
Usage	365
Arguments.....	366
Example	367
Error Messages	369
kmeansplot	369
Summary	369
Usage	369
Example	370
Minhash	371
Summary	371
Background	372
Usage	372
Example	374
Error Messages	376
Canopy	376
Introduction	376
Background	377
Installation.....	377
Driver Usage	377
Arguments	377
Example	377
Error Messages	378

Chapter 8: Naive Bayes.....	380
What is Naive Bayes?	380
Naive Bayes Functions.....	381
naiveBayesMap and naiveBayesReduce	381
Naive Bayes Examples	382
Example Input Data	382
Example SQL-MapReduce call	382
Example Output of Naive Bayes	383
naiveBayesPredict	383
Chapter 9: Decision Trees	386
Random Forest Functions.....	386
Summary	386
Background	387
Usage	388
forest_drive.....	389
Example	392
forest_predict.....	394
Example	396
forest_analyze.....	397
Example	398
Error Messages	398
Best Practices.....	399
Single Decision Tree Functions	400
single_tree_drive	401
Summary	401
Background	401
Usage	403
Example	404
single_tree_predict.....	409
Usage	409
Example	411
Chapter 10: Association Analysis	414
Basket Generator (basket_generator).....	414
Summary	414

Background	414
Usage	415
Examples	416
Collaborative Filtering (cfilter).....	418
Summary	418
Background	418
Usage	418
Example	420
Error Messages	422

Chapter 11: Graph Analysis 424

Overview of Graph Functions.....	424
Graph Types	424
List of Graph Functions	427
nTree	427
Summary	427
Background	428
Usage	428
Examples	431
Support for Very Deep Trees with nTree	434
Using nTree on a pre-5.0 Aster Database.....	435
Single Source Shortest Path (SSSP)	436
Summary	436
Installation.....	436
Syntax (version 1.0)	436
Arguments	437
Input.....	438
Output	438
Example	438
Error Messages	440
degrees (beta).....	441
Summary	441
Background	441
Usage	441
Example	442
triangle_finder (beta).....	444
Summary	444
Background	445
Usage	445
Example	446

rectangle_finder (beta)	448
Summary	448
Background	448
Usage	449
Example	450
local_clustering_coefficient (beta)	452
Summary	452
Background	452
Usage	452
Example	453
pagerank (beta)	454
Summary	454
Background	455
Usage	455
Example	456
eigen_centrality (beta)	458
Summary	458
Background	459
Usage	459
Example	460

Chapter 12: Data Transformation.....	464
Antiselect	464
Summary	464
Background	464
Usage	465
Example	465
Multicase	466
Summary	466
Background	466
Usage	466
Example	467
Pack	469
Summary	469
Usage	469
Example:	470
Unpack	471
Summary	471
Usage	471
Example	473

Pivot	474
Summary	474
Usage	474
Example 1.....	475
Example 2.....	476
Unpivot.....	477
Summary	477
Background	477
Usage	477
Examples	478
Error Messages	480
XMLParser	481
Summary	481
Background	481
Usage	481
Examples	485
Error Messages	489
XMLRelation	490
Summary	490
Background	490
Usage	490
Input Data	491
Output	492
Examples	493
Error Messages	497
JSONParser.....	499
Summary	499
Background	499
Usage	499
Input Data	501
Output	501
Examples	502
Apache Log Parser	505
Summary	505
Background	505
Usage	508
Errors	510
outlierFilter.....	511
Summary	511
Usage	511
IpGeo.....	514
Summary	514

Background	514
Usage	514
Example	515
Extending IpGeo	516

Chapter 13: Visualization Functions..... 522

NpathViz.....	522
Summary and Background.....	522
Usage	525
Input.....	526
Output	527
Examples	527
Error Messages	531
CfilterViz.....	533
Summary and Background.....	533
Usage	533
Input.....	534
Output	534
Examples	535
Visualization Function Usage Notes	538
Directed versus Undirected Graphs	538

Chapter 14: Aster Database System Utility Functions 544

Chapter 15: Aster Lens 546

Aster Lens Concepts.....	547
How Aster Lens Works.....	547
Visualization Tables	547
Visualization Catalog Tables	550
Aster Lens Architecture	551
User Authentication	551
SQL-MR Visualization Functions	551
Installing and Setting Up Aster Lens	552
System Requirements	552
Installing Aster Lens	553
Configuring Aster Lens Properties	555

Encrypting Passwords.....	559
Working with Visualization Catalog Tables.....	559
Installing and Uninstalling Examples	562
Starting and Stopping Aster Lens	563
Uninstalling Aster Lens	563
Manually Installing Aster Lens	564
Installing Aster Lens on the Queen	566
Using Aster Lens.....	569
Displaying Visualizations in Table View.....	570
Displaying the Sidebar	570
Displaying Recent Results	570
Using the Search Field	571
Refreshing the Server Cache.....	571
Logging Out.....	572
Displaying Summary Information	572
Viewing a Visualization	572
Downloading Visualizations	572
Displaying Visualizations By Category	573
Viewing Tree Visualizations.....	573
Viewing Chord Visualizations.....	573
Viewing Sankey Visualizations	573
Viewing Sigma Visualizations	573
Viewing GEXF and Graphviz Visualizations	574
Viewing Log Files	575
Appendix A: List of Functions.....	576
Random Forest Functions	597
Single Decision Tree Functions	598
Index	608

CHAPTER 1 Introduction

- [Analytics at Scale: Full Dataset Analysis](#)
- [Introduction to SQL-MapReduce](#)
- [SQL-MapReduce Query Syntax](#)
- [SQL-MR with Multiple Inputs](#)
- [List of Aster Analytical Functions by Category](#)
- [Aster Analytical Function Descriptions](#)

Analytics at Scale: Full Dataset Analysis

With Aster Database, you have the ability to efficiently perform analytical tasks on your full dataset, in place, rather than using samples or bulk-exporting data to a dedicated computing cluster.

So why run your analytics on your full dataset? While applying analytics to a small sample of the data outside the database might work for some problems, it cannot provide the accurate, reproducible results that can be obtained by analyzing a complete dataset.

One important application of in-database analytics is to speed up iterations of analysis. Since the cycle of iteration time is so critical, many organizations want a solution that is faster than exporting a data sample, analyzing it, and exporting another sample, and so on. In such cases, it makes sense to push down those analytics into an MPP system to decrease the iteration cycle. We are working with partners, including the SAS Institute, Inc., to make this process straightforward, and are additionally developing our functions where appropriate (for example, functionality that really takes advantage of the MapReduce paradigm).

However, there is a stronger set of reasons analyzing the entire data set in Aster Database:

- “*Needle in a Haystack*,” “*False Negative*,” and “*Exceptional Cases*” searches: Very rare events can only be found (and defined) against the background of the entire data set (consider defining ‘elite baseball player’ by looking at the 2008 SF Giants, vs every player in MLB history).
- *Statistical significance*: Reliable analytics may require using a large portion of the data, which cannot be fit on a typical, single database machine.

- *Model tuning:* The parameters to predictive models depend on aggregate statistics of the entire data set (for example, residual away from the mean).
- *No meaningful way to sample:* Sampling a graph is not straightforward, especially if one is interested in critical behavior that only appears when a certain threshold of connections is reached.
- *Larger data sets are just different:* The resulting analytics will be applied to the entire data set in the cluster. Algorithms developed on smaller data sets may not scale appropriately to the full data set, requiring redevelopment.

Introduction to SQL-MapReduce

What is MapReduce?

MapReduce is a framework for operating on large sets of data using MPP (massively parallel processing) systems. The basic ideas behind MapReduce originated with the “map” and “reduce” functions common to many programming languages, though the implementation and application are somewhat different on multi-node systems.

MapReduce enables complex analysis to be performed efficiently on extremely large sets of data, such as those obtained from weblogs, clickstreams, etc. It has applications in areas such as machine learning, scientific data analysis and document classification.

In computer programming languages, a “map” function applies the same operation to every “tuple” (member of a list, element of an array, row of a table, etc.) and produces one output tuple for each input tuple it operates on. It is sometimes called a “transformation” operation. On an MPP database such as Aster Database, the “map” step of a MapReduce function has a special meaning. In this case, the input data is broken up into smaller sets of data, which are distributed to the worker nodes in a cluster, where an instance of the function operates on them. Note that if the data is already distributed as specified in the function call, the distribution step will not occur, because the function can operate on the data where it is already stored. The outputs from these smaller groups of data may be redirected back into the function for further processing, input into another function, or otherwise processed further. Finally, all outputs are consolidated again on the queen to produce the final output result, with one output tuple for each input tuple.

A "reduce" function in computer programming combines the input tuples to produce a single result by using a mathematical operator (like sum, multiply or average). Reduce functions are used to consolidate data or aggregate it into smaller groups of data. They can accept the output of a map function, a reduce function, or operate recursively on their own output.

In Aster Database, the “reduce” step in MapReduce works a little differently, as follows:

- 1 The input data is first be partitioned by the given partitioning attribute.
- 2 The tuples are then distributed to the worker nodes, if required by the function call, with all the tuples that share a partitioning key assigned to the same node for processing.

- 3 On each node, the function operates on these tuples, and returns its output to the queen. Note that the function itself may output the same, a larger or a smaller number of tuples than contained in the input dataset it received.
- 4 The output from each node is consolidated on the queen. Additional operations may be performed on the queen at this time. For example, if the function performs an average, the average results from all the nodes must be averaged again on the queen to obtain the final output.
- 5 The final output is then returned by the SQL-MR function.

Aster Database SQL MapReduce

The Aster Database In-Database MapReduce framework, known as SQL-MapReduce or SQL-MR for short, lets you write functions in Java or C, save these functions in the cluster, and allow analysts to run them in a parallel fashion on Aster Database for efficient data analysis. Analysts invoke a SQL-MR function in a SELECT query and receive the function's output as if the function were a table. A SQL-MR function takes as input one or more sets of rows from tables or views (for example, the contents of a table in the database, the output of a SQL SELECT statement, or the output of another SQL-MR function) and produces a set of rows as output. Beginning in Aster Database 5.0, SQL-MR functions can now accept multiple inputs. For more on this, see “[SQL-MR with Multiple Inputs](#)” on page 29.

Because a call to a SQL-MR function results in a set of parallel tasks being run across the cluster, the input data provided to a SQL-MR function must be divided across the parallel tasks. SQL-MapReduce supports three kinds of inputs:

- 1 We call a SQL-MapReduce function a single input row function if it takes a single row-wise input. When you invoke a row function, you provide it rows in any order. In your SQL statement that calls the row function, you write "ON `my_input`", where `my_input` is your input table. The row function operates at the granularity of individual rows of the `my_input` table. This corresponds to a "map" function in traditional map-reduce systems. The Aster Database SQL-MR API for a function that accepts row-wise input is the *RowFunction* interface.
- 2 We call a SQL-MapReduce function a single input partition function if it takes a single partition-wise input, in which rows are clustered/grouped together by a specified key of one or more columns. In your SQL statement that calls the partition function, you write "ON `my_input` PARTITION BY `partitioning_attributes`" to specify that the function operates on rows sharing a common value of column(s) `partitioning_attributes` of `my_input`; the function has access to all such rows at once, enabling more complex processing than possible with row-wise inputs. Within each partition, you can sort rows using an ORDER BY clause. An Aster Database partition function corresponds to a "reduce" function in traditional map-reduce systems. The Aster Database SQL-MapReduce API for a function that accepts partition-wise input is the *PartitionFunction* interface.
- 3 A SQL-MR function that accepts multiple inputs is called a multiple input function. It can include a cogroup operation in which inputs from multiple sources are partitioned and combined before being processed, a dimension operation where all rows of one or more

inputs are replicated to each vworker, or a combination of both. In your SQL statement that calls the multiple inputs function, you write a combination of the following, to specify each input and how its rows are to be distributed:

- "ON *my_input* PARTITION BY *partitioning_attributes*" for each input where rows are to be partitioned among vworkers using the specified columns,
- "ON *my_input* PARTITION BY ANY" for an input where rows can be processed wherever they were stored when the function was called, and/or
- "ON *my_input* DIMENSION" for each input where all rows are to be replicated to all vworkers.

There are rules governing which types of inputs and how many of each type can be specified in the same multiple input function call. See [Rules for number of inputs by type \(page 32\)](#). The Aster Database SQL-MapReduce API for a function that accepts multiple inputs is the *MultipleInputFunction* interface.

In summary, a SQL-MapReduce function:

- is a function that uses the Aster Database API (Java and C are the supported languages);
- is compiled outside the database, installed (uploaded to the cluster) using Aster Database ACT, and invoked in SQL;
- receives as *input* (from the ON clause(s)) some rows of one or more database tables or views, pre-existing trained models and/or the results of another SQL-MR function;
- receives as *arguments* zero or more argument clauses (parameters), which can modify the function's behavior;
- returns output rows back into the database;
- is polymorphic. During initialization, a function is told the schema of its input (for example, (key, value)) and how it needs to return its output schema;
- is designed to run on a massively parallel system by allowing the user to specify which slice of the data (partition) a particular instance of the function sees.

SQL-MapReduce Query Syntax

Beginning in Aster Database version 5.0, the SQL-MapReduce function syntax is extended to allow one or more partitioned inputs and zero or more dimensional inputs. This has introduced some important changes to the syntax for SQL-MR functions. For more information, see [“SQL-MR with Multiple Inputs” on page 29](#)

Invoking a SQL-MR function has the following syntax in SQL:

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
      * | expression [ [ AS ] output_name ] [, ...]
FROM sqlmr_function_name
      ( on_clause
        function_argument
      ) [ [ AS ] alias ]
[, ...]
[ WHERE condition ]
```

```
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ];
```

where *on_clause* is:

```
partition_any_input | partition_attributes_input | dimensional_input
```

where *partition_any_input* is:

```
table_input PARTITION BY ANY [ORDER BY expression] | table_input [order_by]
```

where *partition_attributes_input* is:

```
table_input PARTITION BY partitioning_attributes [order_by]
```

where *dimensional_input* is:

```
table_input DIMENSION [order_by]
```

where *table_input* is:

```
ON table_expression [ AS alias ]
```

where *table_expression* is:

```
{ table_name | view_name | ( query ) }
```

The synopsis above focuses only on the use of SQL-MapReduce. See SELECT in the *Aster Database User Guide* for a complete synopsis of SELECT, including the WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, and OFFSET clauses.

Notes:

- *sqlmr_function_name* is the name of a SQL-MapReduce function you have installed in Aster Database. If the *sqlmr_function_name* contains uppercase letters, you *must* enclose it in double quotes!
- The *on_clause* provides the *input data* the function operates on. This data is comprised of one or more partitioned inputs and zero or more dimensional inputs. The partitioned inputs can be a single *partition_any_input* clause and/or one or more *partition_attributes_input* clauses. The dimensional inputs can be zero or more *dimensional_input* clauses.
- *partition_any_input* and *partition_attributes_input* introduce expressions that partition the inputs before the function operates on them.

- `partitioning_attributes` specifies the partition key(s) to use to partition the input data before the function operates on it.
- `dimensional_input` introduces an expression that replicates the input to all nodes before the function operates on them.
- `order_by` introduces an expression that sorts the input data after partitioning, but before the function operates on it. It is always optional.
- The `table_input` clause includes an `alias` for the `table_expression`. For rules about when an alias is required, see [Rules for table aliases \(page 32\)](#). When declaring an alias, the `AS` keyword is optional.
- `function_argument` optionally introduces an argument clause that typically modifies the behavior of the SQL-MapReduce function. Don't confuse *argument clauses* with *input data*: Input data is the data the function operates on (see "ON," above), while argument clauses usually just provide runtime parameters. You pass an argument clause in the form `argument_name (literal [, ...])` where `argument_name` is the name of the argument clause (as defined in the function) and `literal` is the value to be assigned to that argument. If an argument clause is a multi-value argument, you can supply a comma-separated list of values. You can pass multiple *argument clause blocks*, each consisting of an `argument_name` followed by its value(s) encased in a single pair of parentheses, separated from the next argument clause block with whitespace (not commas).
- `AS` provides an alias for the SQL-MR function in this query. Using an alias is optional, and, when declaring an alias, the `AS` keyword is optional.

SQL-MR with Multiple Inputs

Beginning in Aster Database version 5.0, Teradata has extended the capabilities of SQL-MR with support for multiple inputs. This allows SQL-MR functions to be applied to related groups of information derived from different data sets. Data from these multiple data sources can be processed within a single SQL-MR function. This changes SQL-MR in two important ways:

- 1 Extending the SQL-MR API to accept multiple inputs allows users to write or modify their own SQL-MR function to do more advanced analysis within the function itself. It essentially mimics the JOIN in SQL, but with better performance.
- 2 The Aster Database Analytics Foundation includes functions that take advantage of the new multiple input capabilities.

Additional changes made to SQL-MR to support this feature are aliasing for table expressions and the addition of syntax to allow specifying explicit partitioning requirements when calling SQL-MR functions.

Benefits of Multiple Inputs

Some benefits of extending SQL-MR to accept multiple inputs are:

- Prediction SQL-MR functions that use a trained model now have better performance and security. The function takes the model itself as a dimensional input, along with one or more data inputs to which it may be applied.
- There is no requirement that all inputs to a SQL-MR function share a common schema.
- The new capabilities avoid JOINS, UNIONs, and the creation of temporary tables, which were often used to work around the older ability to support only a single input.
- New types of analytic functions may now be created more easily (e.g. multichannel attribution).
- Memory is better utilized, because the partitioning and grouping of tuples that occurs before the function operates on them means that less data is actually processed by the function. In addition, the ability to hold one copy of a dimensional input in memory and use it to operate on all tuples from other inputs uses memory more efficiently.

How Multiple Inputs are Processed

When multiple inputs are supplied, SQL-MR performs a grouping on the partitioned inputs, with optional support for dimensional inputs. For functions containing partitioned inputs (PARTITION BY *partitioning_attributes*), the following steps occur:

- 1 The *partitioning_attributes* in all partitioned inputs are examined. A new cogroup tuple is formed for every distinct *partitioning_attributes* that is found. The cogroup tuple's first attribute will be this *partitioning_attributes*.
- 2 For each partitioned input, a new attribute is added to the cogroup tuple. This attribute will hold *all* the attributes of each tuple in that input whose *partitioning_attributes* match the cogroup tuple's *partitioning_attributes*.
- 3 For each dimensional input, a new attribute is added to the cogroup tuple. This attribute contains *all* of that dimensional input's tuples.
- 4 After the above steps occur, we have one cogroup tuple for each distinct *partitioning_attributes* with:
 - one attribute being the *partitioning_attributes*,
 - plus one attribute for each partitioned input that contains a nested array of all of that input's matching tuples,
 - plus one attribute for each dimensional input that includes an array of all of that input's tuples.
- 5 The SQL-MR function then gets invoked on each cogroup tuple.

Comparison semantics are used in this grouping operation, so NULL values are treated as equivalent. Grouped tuples that have empty groups for certain attributes (that is, inputs with no tuples for a particular group) are included in the grouped output by default.

Types of SQL-MR Inputs

From a semantic perspective, there are two possible types of inputs in SQL-MR:

- 1 Partitioned inputs are split up (partitioned) among vworkers as specified in the PARTITION BY clause. These inputs can specify one of:
 - PARTITION BY ANY - random. Note that in practice, PARTITION BY ANY simply preserves any existing partitioning of the data for that input. There may only be one PARTITION BY ANY input in a function.
 - PARTITION BY *partitioning_attributes* - sorted and partitioned on the specified column(s).



Note! All PARTITION BY *partitioning_attributes* clauses in a function must specify the same number of attributes and corresponding attributes must be equijoin compatible (i.e. of the same datatype or datatypes that can be implicitly cast to match). Note that this casting is “partition safe”, meaning that it will not cause any redistribution of data on the vworkers.

- 2 Dimensional inputs use the DIMENSION keyword, and the entire input is distributed to each vworker. This is done because the entire set of data is required on each vworker for the SQL-MR function to run. The most common use cases for dimensional inputs are lookup tables and trained models.

Here’s how it works. A multiple-input SQL-MR function takes as input sets of rows from multiple relations or queries. In addition to the input rows, the function can accept arguments passed by the calling query. The function then effectively combines the partitioned and dimensional inputs into a single nested relation for each unique set of partitioning attributes. The SQL-MR function is then invoked once on each record of the nested relation. It produces a single set of rows as output.

Semantic Requirements for SQL-MR Functions

Keep in mind the following semantic requirements when designing, writing, and calling SQL-MR functions. Note that before applying these rules, Aster Database will assume PARTITION BY ANY for legacy SQL-MR functions whose referencing queries omit the PARTITION BY clause.

What multiple input structures are allowed?

Your multiple-input function always operates on at least two input sets. There are two alternatives for organizing the input data sets:

- You provide the first input set in a row-wise manner and make the other input set(s) available in their entirety. In Aster terminology, we say that the first set is a PARTITION BY ANY set, and the other sets are DIMENSION sets.
- You provide the first input set partitioned on a key you have chosen, and you provide the other input set(s) partitioned on key(s) and/or as DIMENSION set(s).

That’s all you need to know in general about what types of inputs are allowed to work together. The following section lists the specific rules that govern the number and types of inputs that can be used.

Rules for number of inputs by type

In general, any number of input data sets as specified by ON clause constructs can be provided to a SQL-MR function, but the allowed combinations are governed by the rules listed below.

- 1 A function may have at most one PARTITION BY ANY input.
- 2 If a function specifies PARTITION BY ANY, all other inputs must be DIMENSION.
- 3 Any number of grouping attributes as specified by the PARTITION BY *partitioning_attributes* clause can be provided, as long as there are no PARTITION BY ANY inputs.
- 4 All PARTITION BY *partitioning_attributes* clauses must specify the same number of attributes and corresponding attributes must be equijoin compatible (i.e. of the same datatype or of datatypes that can be implicitly cast to match).
- 5 The function will not be invoked if all of the PARTITION BY ANY and PARTITION BY *partitioning_attributes* inputs are empty. Simply having some data in the DIMENSION inputs is not sufficient to invoke the function in itself. DIMENSION inputs are not first class inputs in this sense; they simply come along for the ride as function arguments might.
- 6 The order of the inputs does not matter. For example, your function could have:

```
SELECT ...
ON store_locations DIMENSION,
ON purchases PARTITION BY purchase_date,
ON products DIMENSION ORDER BY prod_name
...
```

Rules for table aliases

- 1 An alias is required for subselects.
- 2 If you are referring to a base table or view, an alias is not required. Aster Database will use the table/view name as the default alias.
- 3 If your SQL-MR function refers to a table or view more than once, then a conflict will be reported, as in this example:

```
SELECT * FROM union_inputs(ON t PARTITION BY ANY ON t DIMENSION mode
('roundrobin'));
```

```
ERROR:  input alias T in SQL-MR function UNION_INPUTS appears more than
once
```

You must give a different alias to each reference to the table or view.

Number of inputs

A SQL-MR function invocation triggers the following validations:

- The multiple input SQL-MR function expects more than one input.
- The single input SQL-MR function expects exactly one input.

Use Cases and Examples for Multiple Inputs

There are many types of SQL-MR functions that can benefit from having multiple data inputs. These generally fall into two classifications: cogroup functions and dimensional input

functions. Note that these are not mutually exclusive; a single function could include both cogroup and dimensional operations.

Cogroup Use Case

Cogroup allows SQL-MR functions to be applied to related groups of information derived from different data sets. The different data inputs are grouped into a nested structure using a cogroup function before the SQL-MR function operates on them.

The following use case is a simplified sales attribution example for purchases made from a web store. We want to find out how much sales revenue to attribute to advertisements, based on impressions (views) and clicks leading to a purchase. As inputs, we have the logs from the web store and the logs from the ad server.

This type of result cannot easily be computed using SQL or SQL-MR capabilities without multiple data inputs.

Cogroup Example

This example uses a fictional SQL-MR function named `attribute_sales` to illustrate how cogroup works. The function accepts two partitioned inputs, as specified in the two ON clauses, and two arguments.

As inputs to the SQL-MR function, we have a `weblog` data set that contains the store's web logs, where we get purchase information. We also have a `adlog` data set that contains the logs from the ad server. We will partition both inputs on the user's browser cookie.

Figure 1: Cogroup Example Tables

Cogroup Example Tables

weblog		
cookie	cart_amt	page
AAAA	\$60	thankyou
AAAA	\$140	thankyou
BBBB	\$100	thankyou
CCCC		intro
CCCCC	\$200	thankyou
DDDD	\$100	thankyou

adlog		
cookie	adname	action
AAAA	champs	impression
AAAA	puppies	click
BBBB	apples	click
CCCC	baseball	impression
CCCC	apples	click

The arguments to the `attribute_sales` function are `clicks` and `impressions`, which supply the percentages of sales to attribute for ad clickthroughs and views (impressions) leading up to a purchase.

We will use the following SQL-MR to call the `attribute_sales` function:

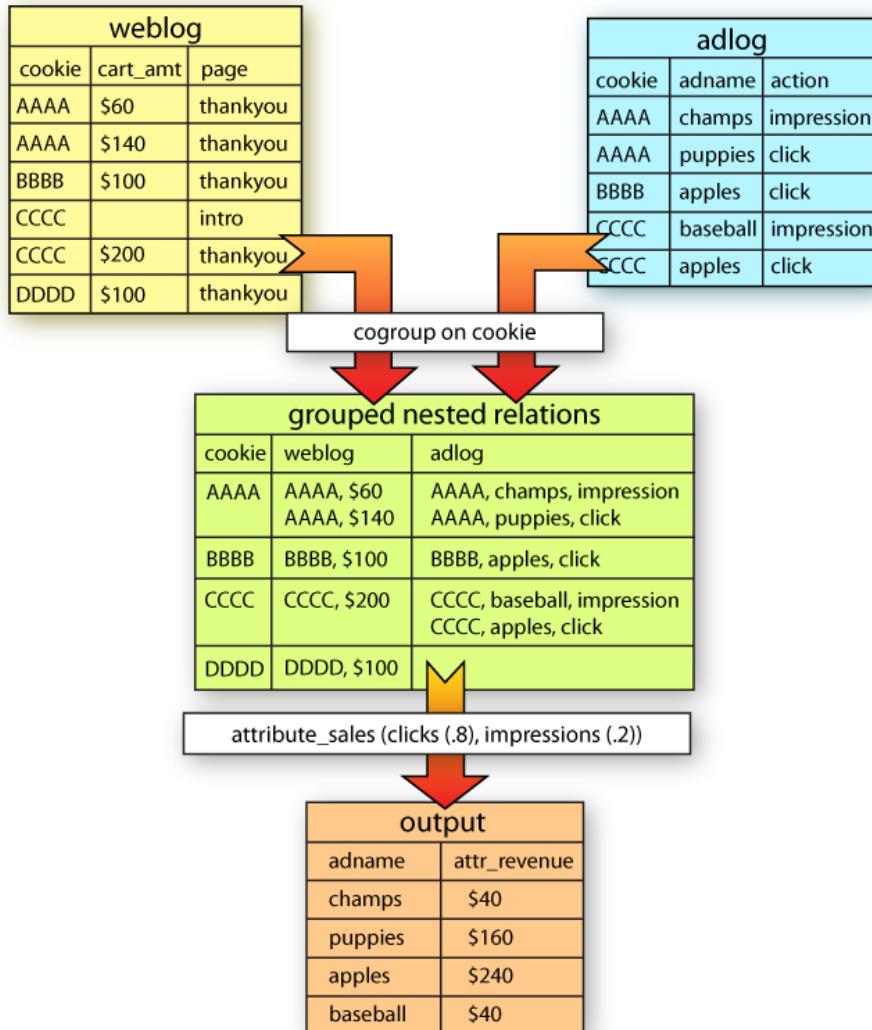
```
SELECT adname, attr_revenue
  FROM attribute_sales (
    ON (SELECT cookie, cart_amt, adname, action
        FROM weblog
       WHERE page = 'thankyou') as W PARTITION BY cookie
    ON adlog as S PARTITION BY cookie
    clicks(.8) impressions(.2))
```

;

The following diagram shows how SQL-MR will execute this function:

Figure 2: How a SQL-MR function performs a cogroup

How Cogroup Works in SQL-MR



The two inputs are cogrouped before the function operates on them. The cogroup operation is conceptually performed in two steps.

- 1 Each input data set is first grouped according to the `cookie` attribute specified in the `PARTITION BY` clauses. A “cogroup tuple” is formed for each unique resulting group. The tuple is comprised of the `cookie` value identifying the group, along with a nested relation that contains all values from both the `weblog` and `adlog` inputs which belong to the group. The middle box in Figure 2 shows the output of the cogroup operation.
- 2 The `attribute_sales` function is then invoked once per cogroup tuple. At each invocation it processes the nested relation, treating it essentially as a single row. The

function then attributes the sales revenue to the appropriate advertisements as previously described. The bottom box in the diagram shows the output of the SQL-MR function.

Note that the cogroup result includes a tuple for the "DDDD" cookie, even though there is no corresponding group in the adlog data set. This is because Aster Database grouping performs an OUTER JOIN, meaning that cogroup tuples that have empty groups for certain attributes are included in the cogroup output.

Dimensional Input Use Case: Lookup Tables

A typical scenario for multiple inputs with dimensional data is when a function needs to access an entire lookup table of values for all rows of a given input. To see how this was accomplished prior to multiple input SQL-MR, let's consider the following example. Suppose a query needed to reference a lookup table of values. Prior to the introduction of multiple inputs, if the lookup table was small enough, one of the following strategies could be used:

- Add the lookup table as an additional row in the query table during the query
- Hold the lookup table in memory for the duration of the query.

There are many scenarios like the following, however, for which the above solutions do not work because of the size, complexity or format of the data:

- The data is too big to fit in memory and/or would make the main query table too large and unwieldy if added to it.
- The data input consists of multi-dimensional data, such as geospatial data.
- The data consist of a model, usually in JSON format, and a set of data to be analyzed against, using the model. For a discussion of this scenario, see [Dimensional Input Use Case: Machine Learning \(page 38\)](#).

In these cases, analysts will find it helpful to use SQL-MR with multiple inputs.

The SQL-MR function can loop over the input data - holding one of the inputs in memory and repeatedly performing the same function on each row of another input. Only one single instance of the dimensional input is held in memory on each of the worker nodes, and it is used in processing each incoming row of partitioned data. Prior to this functionality in SQL-MR, this type of data could not easily be processed. That is because an instance of one of the data inputs had to be held in memory for use by each row of data from any additional input. This could cause slow performance if one or both datasets were very large or of a structure not easily represented in a relational form.

Dimensional Input Example

In this example, we want to create a SQL-MR function to illustrate how dimensional inputs are processed. We'll create the SQL-MR function for a retailer of mobile phone accessories. The function will take data from accessory purchases made by mobile phone and find the closest retail store at the time of purchase. We have two data sets:

- 1 a phone_purchases data set that contains entries for mobile phone accessory purchases along with normalized spatial coordinates of the mobile phone from the time when an online purchase was made, and
- 2 a stores data set that contains the location of all the retail stores and their associated normalized spatial coordinates.

The following diagram shows the two data sets:

Figure 3: Dimensional Example Tables

Dimensional Example Tables

phone_purchases			stores		
pid	xcoord	ycoord	sid	xcoord	ycoord
p0	2	1	s0	1	4
p1	1	5	s1	2	3
p2	3	2			
p3	0	4			

This type of result cannot easily be computed using basic cogroup capabilities, because the data sets need to be related using a proximity join as opposed to an equijoin. [Figure 4](#) illustrates how this is expressed and executed using cogroup extended with dimensional inputs.

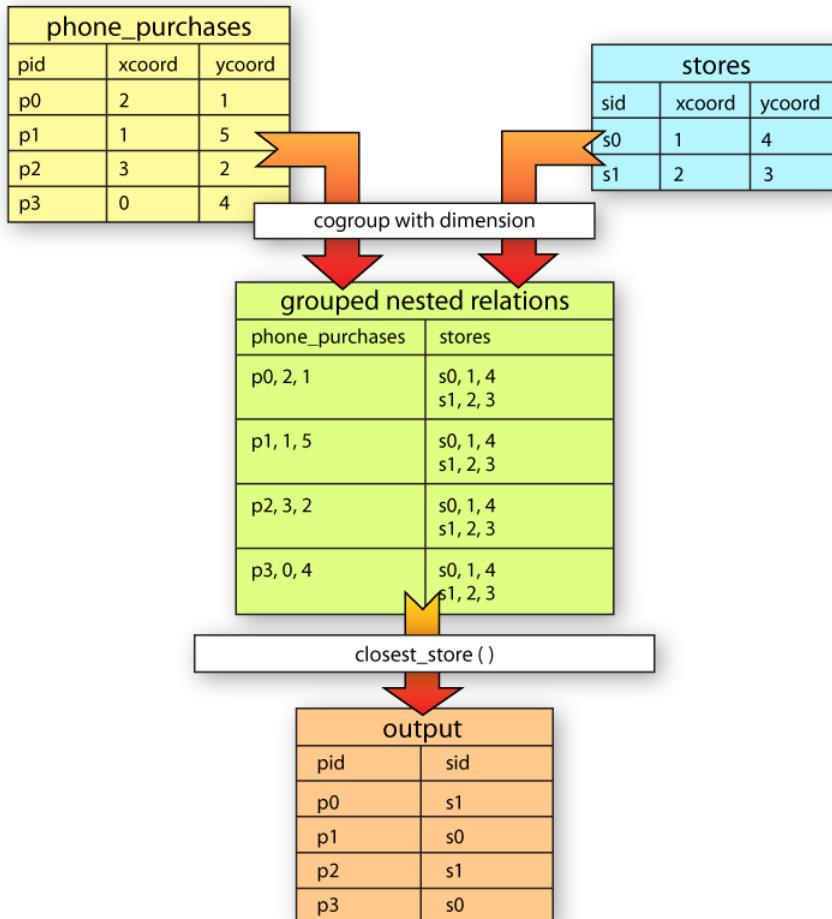
We will create a SQL-MR function named `closest_store`, which accepts two partitioned inputs as specified in two ON clauses. We will use the following SQL-MR to call the `closest_store` function:

```
SELECT pid, sid
  FROM closest_store (
    ON phone_purchases PARTITION BY ANY,
    ON stores DIMENSION)
;
```

The following diagram shows how SQL-MR will execute this function:

Figure 4: How dimensional inputs work in SQL-MR

How Dimensional Inputs Work in SQL-MR



The `closest_store` SQL-MR function receives the result of a cogroup operation on the `phone_purchases` data set and the dimensional input data set `stores`. The two boxes at the top of the diagram show sample `phone_purchases` input and `stores` input respectively. The operation is conceptually performed in three steps:

- 1 The `phone_purchases` input remains grouped in whatever way it is currently stored in the database, as specified by the `PARTITION BY ANY` clause and the `stores` input is grouped into a single group as specified by the `DIMENSION` clause.
- 2 The groups are combined using what is essentially a Cartesian join. The result of the cogroup operation is a nested relation. Each tuple of the nested relation contains (conceptually) an arbitrary group of phone purchases concatenated with the single group comprised all retail stores. The middle box in the diagram shows the result of the cogroup operation.
- 3 The `closest_store` function is subsequently invoked once per cogroup tuple. At each invocation it receives a cursor over an arbitrary group of purchases along with a cursor over the entire set of stores. The function then performs a proximity join using the

normalized spatial coordinates to find the closest store to each purchase. The bottom box in the diagram shows the output of the SQL-MR function.

Dimensional Input Use Case: Machine Learning

Machine learning is another common use case for SQL-MR with multiple inputs. In machine learning, you create or choose a “model” that predicts some outcome given a set of data. You typically test the model to determine its accuracy, fine tuning it until its predictions fall within the desired margin of error. The model itself is comprised of mathematical and statistical algorithms created through observations of patterns found within a given dataset.

Let’s assume you want to generate a trained predictive model, fine tune it, and apply it to a large set of data. Imagine that you have ten million emails, and you need to bucket them by subject matter. You would use a function (such as a “decision tree” function) to generate an algorithm that parses emails and places them in the appropriate bucket. The function might do some statistical analysis to determine where clusters of data appear, and create subject matter “buckets” based on these. These emails might be placed into buckets based on frequency of occurrence of certain words, work proximity and/or grammatical analysis.

To test the accuracy of your model, you might have a human being do this same classification work manually for a subset of the emails (called the "sample dataset"). The sample dataset might be one thousand emails. The person would read and classify each email according to the desired criteria. The model is then applied to the sample data set and the results compared to the known outcome (the results generated by the human being). You can then gauge the reliability of the predictive model and fine tune it. Finally, you can do the analysis on future emails using the predictive model, with a known margin of error.

Some functions that can generate these models include Naive Bayes, K-means nearest neighbor, decision trees, and logistic regression. The model generated by this type of function does not generally follow a relational structure. It is more likely to be in a JSON format. It can be stored in the file system, but it is more commonly stored in a database.

So in the machine learning scenario, the data inputs to the SQL-MR function will consist of 1) a model, usually in JSON format, and 2) one or more sets of data to be analyzed against, using the model. Similar to the lookup table example, the predictive model must be applied to each row of input from the new data set. Thus, the predictive model will be input to the function using the DIMENSION keyword. The data to be analyzed could use either PARTITION BY ANY or PARTITION BY *partitioning_attributes*.

SQL-MR Multiple Input FAQ

How are multiple inputs combined?

Multiple inputs are combined using what is essentially a cogroup operation, with the addition of support for dimensional inputs. Grouping is done using an OUTER JOIN where NULLs compare equal. The SQL-MR function is effectively invoked once for each unique partition of all the partitioned inputs. All dimensional inputs are provided at each invocation. The function can output one or more tuples at each invocation.

There are two mutually exclusive cases to consider in determining what the unique partitions of the partitioned inputs will be:

- One or more `partition_attributes_input` inputs are combined into partitions using a cogroup operation. The cogroup operation forms one partition for each unique combination of partitioning attributes present in any of the inputs. Each partition provides the values of the partitioning attributes and the tuples from each input that agree on those values. If a given input has no tuples for a particular combination of partitioning attributes, then an empty set of tuples is provided for that input.
- A single `partition_any_input` is processed wherever its data is stored. Each invocation provides the input tuples to the vworker where they currently reside in Aster Database.

Note that the function is not invoked if all of the inputs tables to partitioned inputs are empty, even if a dimensional input has been provided. Thus the dimensional inputs are not first class inputs in that they do not drive invocation of the function. They are simply provided as additional input to the function.

Can dimensional inputs include non-deterministic expressions?

No. Dimensional inputs should not include non-deterministic expressions. These are expressions that are not guaranteed to evaluate to the same result every time (expressions with a volatility other than IMMUTABLE). Note that whenever changing Global User Configuration (GUC) settings can change the result of an expression, the volatility of that expression may be classified as STABLE, but it cannot be classified as IMMUTABLE. An example of this would be changing Locale and Formatting settings, such as datestyle or time zone.

Where will the output of the function be located?

The output location for row and cogroup functions is determined by the function:

- If it is vworker specific, the output goes to that worker.
- If it is partitioned, the output will be partitioned in the same way.
- If it is replicated, the output will be replicated.

If the input is located on the queen:

- If the inputs are row or dimension inputs, the output will be replicated to the vworkers.
- If the input is partitioned, the output will also be partitioned.

List of Aster Analytical Functions by Category

This section lists the names of the Aster Analytical functions and their categories. For a list of functions and their syntax, see “[List of Functions](#)” on page 576.

Table 1 - 1: Functions by Category

Category	Functions
Time Series, Path, and Attribution Analysis	<ul style="list-style-type: none"> • Cumulative Moving Average • SAX • Path Generator • Path Starter • Path Summarizer • Sessionization • Attribution • FrequentPaths • DWT • IDWT • DWT2d • IDWT2d
Pattern Matching with Teradata Aster nPath™	<ul style="list-style-type: none"> • nPath
Statistical Analysis	<ul style="list-style-type: none"> • Approximate Distinct Count (count_approx_distinct) • Approximate Percentile (approx_percentile) • Correlation (stats correlation) • Histogram • Enhanced Histogram Function • Linear Regression (stats linear reg) • Logistic Regression (deprecated) • Generalized Linear Model (stats glm) • Generalized Linear Model Prediction (glmpredict) • Principal Component Analysis (PCA) • Simple Moving Average (stats smavg) • Weighted Moving Average (stats wmavg) • Exponential Moving Average (stats emavg) • Volume-Weighted Average Price (stats vwap) • K-Nearest Neighbor Classification Algorithm (knn) • Support Vector Machines • ConfusionMatrix • Percentile • Distribution Matching • LARS Functions • Sample • FMeasure

Introduction

List of Aster Analytical Functions by Category

Table 1 - 1: Functions by Category (continued)

Category	Functions
Text Analysis	<ul style="list-style-type: none">• Levenshtein Distance• nGram• Text Classifier• Text Parser (text_parser)• Named Entity Recognition (NER)• FindNamedEntity• TrainNamedEntityFinder• EvaluateNamedEntityFinderRow and EvaluateNamedEntityFinderPartition• Sentiment Extraction Functions ExtractSentiment EvaluateSentimentExtractor TrainSentimentExtractor• Naive Bayes Text Classifier Naive Bayes Text (naiveBayesText) Naive Bayes Text Predict (naiveBayesTextPredict)
Cluster Analysis	<ul style="list-style-type: none">• kmeans• kmeansplot• Minhash• Canopy
Naive Bayes	<ul style="list-style-type: none">• naiveBayesMap and naiveBayesReduce• naiveBayesPredict
Decision Trees	<ul style="list-style-type: none">• Random Forest Functions forest_drive forest_predict forest_analyze• Single Decision Tree Functions single_tree_drive single_tree_predict
Association Analysis	<ul style="list-style-type: none">• Basket Generator (basket_generator)• Collaborative Filtering (cfilter)
Graph Analysis	<ul style="list-style-type: none">• nTree• Single Source Shortest Path (SSSP)• degrees (beta)• triangle_finder (beta)• rectangle_finder (beta)• local_clustering_coefficient (beta)• pagerank (beta)• eigen_centrality (beta)

Table 1 - 1: Functions by Category (continued)

Category	Functions
Data Transformation	<ul style="list-style-type: none">• Antiselect• Multicase• Pack• Unpack• Pivot• Unpivot• XMLParser• XMLRelation• JSONParser• Apache Log Parser• outlierFilter• IpGeo
Visualization Functions	<ul style="list-style-type: none">• NpathViz• CfilterViz

Aster Analytical Function Descriptions

Below, we list the Aster SQL-MapReduce analytical functions, grouped by the type of analysis or utility function that each handles.

Time Series, Path, and Attribution Analysis

Table 1 - 2: Time Series, Path, and Attribution Analysis Functions

Function	Description
Cumulative Moving Average	The Cumulative Moving Average (CMAVG) function computes the average of a value from the beginning of a series.
SAX	Symbolic Aggregate approXimation (SAX) transforms original time series data into symbolic strings. Once this transformation is complete, the data is more suitable for many additional types of manipulation, both because of its smaller size and the relative ease with which patterns can be identified and compared.
Path Generator	This function takes as input a set of paths where each path is a route (series of pageviews) taken by a user from start to end. For each path, it generates the correctly formatted sequence and all possible sub-sequences for further analysis by the Path Summarizer function. The first element in the path is the first page a user could visit. The last element of the path is the last page visited by the user.
Path Starter	Generates all the children for a particular parent and sums up their count. Note that the input data has to be partitioned by the parent column.
Path Summarizer	The output of the Path Generator function is the input to this function. This function is used to sum counts on nodes. “Node” can either be a plain sub-sequence or an exit sub-sequence. Exit sub-sequence is the one in which both sequence and the sub-sequence are same. Exit sub-sequences are denoted by appending '\$' to the end of the sequence.
Sessionization	Sessionization is the process of mapping each click in a clickstream to a unique session identifier. One can define a session as a sequence of clicks by a particular user where no more than n seconds pass between successive clicks (that is, if we don't see a click from a user for n seconds, we start a new session).
Attribution	The attribution operator is often used in web page analysis. Companies would like to assign weights to pages before certain events, such as a 'click' or a 'buy'. This attribution function enables you to calculate attributions by using a wide range of distribution models.
FrequentPaths	Mines for patterns that appear more than a certain number of times in the sequence database. The difference between sequential pattern mining and frequent pattern mining is that the former works on time sequences where the order of items must be kept.
DWT	Implements Mallat's algorithm, which is an iterate algorithm in the Discrete Wavelet Transform (DWT) field, and is designed to apply wavelet transform on multiple sequences simultaneously.
IDWT	Applies inverse wavelet transformation on multiple sequences simultaneously. IDWT is the inverse of DWT.
DWT2d	Implements wavelet transforms on two-dimensional input, and simultaneously applies the transforms on multiple sequences.

Table 1 - 2: Time Series, Path, and Attribution Analysis Functions (continued)

Function	Description
IDWT2d	Simultaneously applies inverse wavelet transforms on multiple sequences. IDWT2d is the inverse function of DWT2d.

Pattern Matching with Teradata Aster nPath

Table 1 - 3: Pattern Matching with Teradata Aster nPath

Function	Description
nPath	Teradata Aster nPath is a function for pattern matching that allows you to specify a pattern in an ordered collection of rows, specify additional conditions on the rows matching these symbols, and extract useful information from these row sequences.

Statistical Analysis

Table 1 - 4: Statistical Analysis

Function	Description
Approximate Distinct Count (count_approx_distinct)	Computes an approximate global distinct count of the values in the specified column or combination of columns. Based on probabilistic counting algorithms, this algorithm counts the approximate distinct values for any number of columns or combination of columns, while scanning the table only once. Evaluates all the children for a particular parent and sums up their count. Note that the input data has to be partitioned by the parent column.
Approximate Percentile (approx_percentile)	Computes approximate percentiles for one or more columns of data where the accuracy of the approximation is a parameter specified by the user.
Correlation (stats correlation)	Computes a global correlation between any pair of columns from a table.
Histogram	Counts the number of occurrences of a given data value that fall into each of a series of user-defined bins.
Enhanced Histogram Function	Adds new capabilities over the existing histogram function (represented by histogram_map and histogram_reduce).
Linear Regression (stats linear reg)	Outputs the coefficients of the linear regression model represented by the input matrices.
Logistic Regression (deprecated)	A series of row functions and partition functions that establish the weights sequence for the logistic regression.
Generalized Linear Model (stats glm)	GLM performs linear regression analysis for any of a number of distribution functions using a user-specified distribution family and link function. Supported models in Aster Database are ordinary linear regression, logistic regression (logit model), and Poisson log-linear model.
Generalized Linear Model Prediction (glmpredict)	Scores input data using the model generated by the Stats GLM function.
Principal Component Analysis (PCA)	Principal component analysis (PCA) is a common unsupervised learning technique that is useful for both exploratory data analysis and dimensionality reduction. It is often used as the core procedure for factor analysis.
Simple Moving Average (stats smavg)	Computes the average over a number of points in a series.

Introduction

Aster Analytical Function Descriptions

Table 1 - 4: Statistical Analysis (continued)

Function	Description
Weighted Moving Average (stats wavg)	Computes the average over a number of points in a time series while applying an arithmetically-decreasing weighting to older values.
Exponential Moving Average (stats emavg)	Computes the average over a number of points in a time series while applying an exponentially decaying damping (weighting) factor to older values so that more recent values are given a heavier weight in the calculation.
Volume-Weighted Average Price (stats vwap)	Computes the average price of a traded item (usually an equity share) over a specified time interval.
K-Nearest Neighbor Classification Algorithm (knn)	Uses the <i>k</i> NN algorithm to classify data objects based on their proximity to training objects with known classification.
Support Vector Machines	Use this function to generate a model that can be used by the svm_predict function to predict data classification.
ConfusionMatrix	Defines a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.
Percentile	Finds percentiles on a per group basis.
Distribution Matching	Carries out hypothesis testing and finds the best matching distribution for the data.
LARS Functions	Select most important variables one by one and fit the coefficients dynamically. The LARS function implements a model selection algorithm. LarsPredict takes in the new data and the model generated by LARS, and outputs the predictions.
Sample	Draws rows randomly from the input relation. The function offers two sampling schemes.
FMeasure	Calculates the accuracy of a test.

Text Analysis

Table 1 - 5: Text Analysis

Function	Description
Levenshtein Distance	Computes the Levenshtein distance between two text values, that is, the number of edits needed to transform one string into the other, where edits include insertions, deletions, or substitutions of individual characters.
nGram	Tokenizes (or splits) an input stream and emits n multi-grams based on the specified delimiter and reset parameters. This function is useful for performing sentiment analysis, topic identification, and document classification.
Text Classifier	Chooses the correct class label for a given text input.
Text Parser (text_parser)	A general tool for working with text fields that can tokenize an input stream of words, optionally stem them, and then emit the individual words and counts for the each word appearance.
Named Entity Recognition (NER)	Named entity recognition (NER) is a process of finding instances of specified entities in text (For example, person, location, organization, etc.) It has functions to train, evaluate and apply models which perform this analysis.

Table 1 - 5: Text Analysis (continued)

Function	Description
Sentiment Extraction Functions	The sentiment extraction functions enable the process of deducing a user's opinion (positive, negative, neutral) from text-based content.
Naive Bayes Text Classifier	Determines the classification of data objects based on the Naive Bayes algorithm, which takes into account the classification probability based on the training data set and additional input variables.

Cluster Analysis

Table 1 - 6: Cluster Analysis

Function	Description
kmeans	Simple unsupervised learning algorithm that solves the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The goal is to define k centroids, one for each cluster.
kmeansplot	A function that clusters new data points around the cluster centroids generated by the k-Means function.
Minhash	A probabilistic clustering method that assigns a pair of users to the same cluster with probability proportional to the overlap between the set of items that these users have bought (this relationship between users and items mimics various other transactional models).
Canopy	A simple, fast, accurate method for grouping objects into preliminary clusters. Each object is represented as a point in a multidimensional feature space. Canopy clustering is often used as an initial step in more rigorous clustering techniques, such as k-means clustering.

Naive Bayes

Table 1 - 7: Naive Bayes

Function	Description
naiveBayesMap and naiveBayesReduce naiveBayesPredict	The Naive Bayes Functions let you train a Naive Bayes classification model, and use the model to predict new outcomes. Observed data from a sample set with known outcomes is input to the function. The function then creates a model that can later be applied to observed or hypothetical data where the outcome is not known. Predicted outcomes with statistical probability are then determined and output by the model.

Decision Trees

Table 1 - 8: Decision Trees

Function	Description
forest_drive	The Random Forest Functions let you create a predictive model based on a combination of the CART algorithm for training decision trees, and the ensemble learning method of bagging.
forest_predict	
forest_analyze	
single_tree_drive	These Single Decision Tree Functions let you create a predictive model without creating multiple decision trees. Only one decision tree is created.
single_tree_predict	

Association Analysis

Table 1 - 9: Association Analysis

Function	Description
Basket Generator (basket_generator)	Generates sets or “baskets” of items that occur together in records in data, typically transaction records or web page logs.
Collaborative Filtering (cfilter)	Helps discover which items or events are frequently paired with other items or events.

Graph Analysis

Table 1 - 10: Graph Analysis

Function	Description
nTree	nTree is a hierarchical analysis SQL-MR function which can build and traverse through tree structures on all worker machines.
Single Source Shortest Path (SSSP)	Given a graph with vertices and edges between these vertices, the Single Source Shortest Path (SSSP) function finds the shortest paths from a given vertex to all the other vertices in the graph.
degrees (beta)	This function generates the in-degree and out-degree tables for a directed graph. The degrees function is the simplest way to calculate the centrality measure of a graph.
triangle_finder (beta)	This function finds triangles in an undirected graph. Triangles, or 3-cycles, can be the basis for analyzing graphs and subgraphs. In particular, enumerating triangles constitutes most of the work in identifying the dense subgraphs called trusses.
rectangle_finder (beta)	This function finds rectangles in an undirected graph. The job of enumerating rectangles (4-cycles) is similar to that of enumerating triangles.
local_clustering_coefficient (beta)	This function computes the local clustering coefficient for nodes whose local clustering coefficient is not zero in an undirected graph.
pagerank (beta)	PageRank is a link analysis algorithm. It assigns a numerical weighting (between 0 and 1) to each node in a graph, with the purpose of <i>measuring</i> its relative importance within the graph.

Table 1 - 10: Graph Analysis (continued)

Function	Description
eigen_centrality (beta)	Eigenvector centrality is a measure of the influence of a node in a network. An algorithm implementing eigenvector centrality assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.

Data Transformation

Table 1 - 11: Data Transformation

Function	Description
Antiselect	Returns all columns except the columns specified.
Multicase	Extends the capability of the SQL CASE statement by supporting matches to multiple options. The function iterates through the input data set only once and emits matches whenever a match occurs whereas as soon as CASE has a match it emits the result and then moves on to the next row.
Pack	Compress data in multiple columns into a single “packed” data column.
Unpack	Take data from a single “packed” column and expand it to multiple columns.
Pivot	Pivots data stored in rows into columns.
Unpivot	Converts columns into rows.
XMLParser	Extracts the data from XML documents. and flatten it into a relational table.
XMLRelation	The XMLRelation function is a tool for extracting most XML content (element name, text and attribute values) and structural information from XML documents into a relational table.
JSONParser	The JSONParser function is a tool used to extract the element name and text from JSON strings and output them into a flattened relational table.
Apache Log Parser	Parses Apache log file content and extracts multiple columns of structural information, including search engines and search terms.
outlierFilter	Removes outliers from their data set.
IpGeo	IpGeo lets you map IP addresses to location information. You can use this information to identify the geographical location of a visitor. This information includes country, region, city, latitude, longitude, ZIP code, and ISP.

Visualization Functions

Table 1 - 12: Visualization Functions

Function	Description
NpathViz	Visualizes the output of the Teradata Aster nPath function.
CfilterViz	Visualizes the output of the cfilter function.

Aster Database Utilities

For information on the functions used for querying data from local vworkers, “[Aster Database System Utility Functions](#)” on page 544. These functions are useful for querying local catalog tables to support database administration activities.

CHAPTER 2 Installing Analytical Functions

- [Aster Analytical Function Versioning](#)
- [Installation Procedure](#)
- [Using the \install Command in ACT](#)
- [Installing Aster Database Driver-Based Analytical Functions](#)
- [Using Double-Quotes in Database Object Names](#)
- [Creating a Timestamp Column](#)
- [Best Practices](#)
- [Error Message Delays](#)

Aster Analytical Function Versioning

Beginning in Aster Analytics 5.0-Release 1, Aster Analytical Functions have a version number assigned by Teradata Aster. The version number is read-only, and it is provided to help you:

- 1 check whether the installed function is compatible with your version of Aster Database.
- 2 ensure that the documentation you are using matches the version number of the installed function.
- 3 verify that any function updates have been successful.

Whenever a new Analytics Foundation is released, it is recommended to upgrade all functions. To review any changes made to existing SQL-MR function syntax by the upgrade, review the function in this guide and the Release Notes for your version of Aster Analytics Foundation.

The function version number may be viewed by issuing the command `\dE` in ACT, which returns results like:

List of SQL-MR functions					
schemaname	funcname	funcowner	funcversion	creationtime	
nc_system	load_from_hcatalog	db_admin		2012-09-24 20:36:13.781095	
...					
nc_system	nc_tablesize_details	db_admin		2012-09-24 20:36:13.781095	
nc_system	stream	db_admin		2012-09-24 20:36:13.781095	
public	antiselect	beehive	5.0_rel_1.0_r29904	2012-09-26 07:09:17.14943	
public	apache_log_parser	beehive	5.0_rel_2.0_r29904	2012-09-26 07:10:32.966452	
...					
public	xmlexporter	beehive	5.0_rel_1.1_r29904	2012-09-26 07:10:33.624989	
public	xmlrelation	beehive	5.0_rel_1.0_r29904	2012-09-26 07:10:36.220648	
(134 rows)					

The function version number may also be viewed by querying one of the following system tables:

- `nc_user_sqlmr_funcs` lists installed functions to which the current user has EXECUTE privileges.
- `nc_user_owned_sqlmr_funcs` lists installed functions owned by the current user.
- `nc_all_sqlmr_funcs` lists all installed functions in the database. This view is only viewable by the administrator.

You can do this from ACT, by issuing a command similar to:

```
SELECT funcname, funcversion from nc_user_sqlmr_funcs;
```

Which returns a result like:

funcname	funcversion
stream	
nc_genericlocalquery	
nc_tablesize	
nc_tablesize_details	
nc_skew	
nc_recursive	
load_from_teradata	
load_to_teradata	
load_from_hcatalog	
approx_dcount_combine	5.0_rel_1.0_r29904
approx_dcount_partial	5.0_rel_1.0_r29904
...	
textclassifierevaluator	5.0_rel_1.0_r29904
textclassifier	5.0_rel_1.0_r29904
(135 rows)	

The function version displayed will be in the format:

```
<point_release_number>_rel_<function_version>_<build_number>
```

Here is an example with the function version highlighted:

```
5.0_rel_1.0_r29904
```

Note that the utility functions and connector functions do not display a version number.

In this guide, the version number for each function appears in the “Syntax” section, just before the function syntax. When using this guide for function syntax, ensure that the installed version of the function matches the documentation.

Installation Procedure

- [Download the Analytics Foundation Bundle](#)
- [Install the Functions](#)
- [Upgrade the Functions](#)
- [Set Permissions to Allow Users to Run Functions](#)
- [Test the Functions](#)

Download the Analytics Foundation Bundle

The fastest way to install the Aster SQL-MapReduce analytical functions is to run the `install.sql` script provided in each Analytics Foundation bundle. Follow these steps:

- 1 Download the desired version of the Analytics Foundation bundle for the functions you have purchased. The file name for the Analytics Foundation bundle depends on which package of analytical functions you have purchased, but the name will be similar to this: “`Analytics_Foundation.zip`”.
- 2 Unpack the zip archive on the machine where you run ACT. You will use ACT to install the functions from the bundle. On Windows, use a tool such as *WinRAR* to extract the archive’s contents. On Linux, use a tool such as *Info-Zip unzip*.

For this example, we will assume you have extracted the zip’s contents to `C:\analyticslibs` on a Windows machine.

Next Step

Proceed to “[Install the Functions](#)” on page 52, below.

Download Beta Analytic Functions

To download Aster Analytics beta SQL-MR functions, go to:

<http://downloads.teradata.com/download/aster>

This page includes a section for downloading beta functions.

Next Step

Proceed to “[Install the Functions](#)” on page 52, below.

Install the Functions

The following instructions are only for installing new functions. To upgrade existing functions, see “[Upgrade the Functions](#)” on page 53.

- 1 Working at the command line, change to the directory where you extracted the files.
- 2 Run ACT, logging in with an Aster Database user account that has rights to install functions in the *public* schema.

Ideally, your account should also have rights to grant EXECUTE and other permissions on the functions you install.

Note: Aster Database does not share database objects, including functions, across databases.

```
$ act -h <Queen IP Address> -U db_superuser  
Password: *****
```

- 3 In ACT, use the `\i` command to run the `install.sql` script:

```
beehive=>\i install.sql
```

The script installs the set of functions you purchased.

Note: If you would like to install some but not all of the functions from your package, see the section below, “[Using the \install Command in ACT](#)” on page 54.

- 4 Type \dF to review the list of installed functions.

Next Step

Proceed to “[Set Permissions to Allow Users to Run Functions](#)” on page 53, below.

Upgrade the Functions

To install an update to an existing SQL-MR function in Aster Database:

- 1 Remove any existing installed version of the function using the \remove command in ACT:

```
\remove <function_filename>
```

- 2 Then install the new version:

```
\install <function_filename>
```

To install and update all of the functions in a new Analytics Foundation release:

- 1 Run the script from the un_install.sql file (found in the Analytics_Foundation.zip file) in ACT by issuing the following command:

```
act -f un_install.sql
```

- 2 Then run the install script:

```
act -f install.sql
```

You may, however, maintain a separate version of a function in a separate database schema. Instructions for this are included in the section “[Installing a function in a specific schema](#)” on [page 56](#).

Next Step

Proceed to “[Set Permissions to Allow Users to Run Functions](#)” on page 53, below.

Set Permissions to Allow Users to Run Functions

- 1 Type \dF <function name> to check which schema the functions belong to.
- 2 Use the GRANT command to give the EXECUTE privilege to users who will run each function. The syntax is:

```
GRANT EXECUTE  
    ON FUNCTION <schema-name>.<function-name>  
    TO <user-name or group-name or PUBLIC>;
```

For example, to give user *beehive* the right to run the function *path_start.jar*, you would type:

```
GRANT EXECUTE  
    ON FUNCTION public.path_start  
    TO beehive;
```

Note that in most ACT commands for managing functions, when you type the function name, *you do not type its suffix* (like “.jar” in this example).

- 3 Repeat the preceding step for all functions and users. To quickly grant broad access, grant the EXECUTE privilege on each function to PUBLIC.

To learn more about the rules that govern who can install and run functions in Aster Database, see the section, “SQL-MapReduce Security” in the *Aster Database User’s Guide*.

Test the Functions

After you have performed the preceding steps, the function is installed and usable by all users to whom you’ve granted EXECUTE rights. Test your functions by following the steps below to run them:

- 1 Run Aster Database ACT and log in as an SQL user who has the EXECUTE privilege on a function.
- 2 Invoke the function in a statement such as a SELECT or other data-retrieval statement. Make sure you schema-qualify the function’s name, or have its schema in your schema search path.

Using the \install Command in ACT

You can install and manage individual Aster SQL-MapReduce Analytics functions, SQL-MapReduce functions, *stream()* functions, and other installed files using the \install command and related commands in the *Aster Database* ACT tool. (Note that if you’re installing all the analytics functions, it’s faster to use the *install.sql* script as explained in “[Download the Analytics Foundation Bundle](#)” on page 52.)

Using \install, you can install:

- SQL-MapReduce functions: Compiled Java and C executables that can be called by name in the FROM clause of a SELECT.
- Scripts for *stream()*: Each script is installed as a file that you will invoke in a call to *stream()*.
- Files: Installed files are typically used to provide settings to SQL-MapReduce functions and to *stream()* functions. Installed files can only be used in your SQL-MapReduce and *stream()* functions. Installed files are not directly invokable or accessible via SQL.

The ACT commands for installing and removing files and functions are listed below. Here, we refer to a file or function as “local” when it resides on your local file system, and as “remote” when it resides in Aster Database.

Table 2 - 13: ACT commands for managing files and functions

Command	Meaning
\dF	Lists all installed files and functions.
\install <i>file</i> [<i>installed_filename</i>]	Installs the file or function called <i>file</i> . The argument, <i>file</i> , is the path name of the file relative to the directory where ACT is running. Optionally, you can give the file or function an <i>installed_filename</i> alias. Aliases are provided as a convenience that's mainly useful for renaming helper files you install. Using an alias for an SQL-MapReduce function can be confusing, so we don't recommend doing it. If no <i>installed_filename</i> is specified, the file's name will be used as its name in Aster Database. Keep in mind that, when you call an SQL-MapReduce function in your queries, you drop its filename suffix. If the file or function does have an <i>installed_filename</i> , then all calls to it from other functions or from queries must use its <i>installed_filename</i> .
\download <i>installed_filename</i> [<i>newfilename</i>]	Downloads the specified, installed file or function (identified by its <i>installed_filename</i>) to the machine where ACT is running. Optionally, you can specify a new name for the file by supplying the <i>newfilename</i> argument. This argument can be a path, but the destination directory must exist on the file system where you're running ACT.
\remove <i>installed_filename</i>	Removes the file or function specified by its FILE_ALIAS.

The `install` and `\remove` commands can be used transactionally in a BEGIN / COMMIT block just like any transactional SQL command.

Tip! You should only install files in the *public* schema of your database. The only reason to install some functions in a separate schema might be that you need to maintain multiple versions of specific functions for backwards compatibility. To do that, see

Installing a function in a specific schema

Generally, the best practice is to install SQL-MR functions only in the public schema. However, if for some reason you need to maintain a separate instance of a function (for example, an older version for compatibility with existing scripts), you may install functions in separate schemas. To install functions in a schema other than public, prepend the function filename with <schemaname>/ using commands like:

```
\install counttokens.jar textanalysis/counttokens.jar
\install tokenize.jar textanalysis/tokenize.jar
```

Then, to use call the functions in that schema, you can issue:

```
SELECT token, count FROM textanalysis.counttokens
  (ON (SELECT token, count FROM textanalysis.tokenize
        (ON documents) PARTITION BY token
      ) ;
```

Installing Aster Database Driver-Based Analytical Functions

Some Aster SQL-MapReduce analytical functions are packaged as driver-based applications that connect to the database over JDBC. These functions can be thought of as stored procedures, but with more capability for developing logical applications. These JDBC programs typically consist of two components: a JDBC client-function and one or more SQL-MapReduce functions. The JDBC client-function executes on the client machine (which can be of any platform). The SQL-MapReduce functions reside in a database in your Aster Database cluster, and they are executed on the cluster. The JDBC client-function interacts with the SQL-MapReduce functions over the network.

This section shows you how to install and set up the components you need in order to use driver-based analytical functions.

- [Required Components](#)
- [Installing the SQL-MapReduce Functions in Aster Database](#)
- [Installing the JDK on the Client System](#)
- [Installing the Aster SQL-MapReduce API on the Client System](#)
- [Testing the Installation of a JDBC-Based Analytics Function](#)

Required Components

To run a JDBC-based Aster Analytics function, the function's components must be installed in your cluster, and on your client machine you must have:

- the JDK (Java SE Development Kit) version 5 or later
- the Aster Database JDBC driver
- the Aster SQL-MapReduce API library

Below, we will show you how to set these up.

Some of the functions are standalone functions, and others are made up of several different functions. You can find the list of functions required for each function in its description in this document.

Installing the SQL-MapReduce Functions in Aster Database

If you have already installed all the SQL-MapReduce functions by running the install script `install.sql`, you may skip this step and proceed to **Step 1** in the next section. Otherwise:

- 1 Install the desired Aster Database SQL-MapReduce functions:

- a Run ACT, connecting as a database administrator who has the INSTALL FILE privilege and the CREATE FUNCTION privilege in the schema where you will install the function:

```
# act -h 10.50.52.100 -U db_superuser
```

- b Run the \install command, passing the name of the function to be installed:

```
\install <function_name>.<jar/zip>
```

For example, to use the Single-Source Shortest Path (SSSP) function, you must install three SQL-MapReduce functions on your cluster. These functions can be installed using the following commands in ACT:

```
beehive=> \install sssp_prepare.jar
beehive=> \install sssp_map.jar
beehive=> \install sssp_reduce.jar
```

Installing the JDK on the Client System

- 1 Download the JDK from <http://java.sun.com/javase/downloads/index.jsp> and
- 2 Install the JDK on your client machine according to the JDK installation instructions at <http://java.sun.com/javase/6/webnotes/install/index.html>

Installing the Aster Database JDBC driver on the client system

- 1 Get the Aster Database JDBC driver, noarch-ncluster-jdbc-driver.jar, in one of these ways:
 - Copy the package from your queen node. On the queen, you can find the installers in `/home/beehive/clients_all/platform` (where *platform* is the name of your client machine's operating system)
 - Download the driver from <http://downloads.teradata.com/download/tools>
 - Copy the driver to a location in your CLASSPATH on the client machine, or edit the client machine's CLASSPATH to include its directory.

Installing the Aster SQL-MapReduce API on the Client System

- 1 Download the Aster SQL-MapReduce API, ncluster-sqlmr-api.jar, from the queen to your client machine. This library can be found on your Aster Database queen in the directory, saved as `/home/beehive/bin/lib/sqlmr/ncluster-sqlmr-api.jar`
- 2 Copy the SQL-MapReduce API library to a location in your CLASSPATH on the client machine, or edit the client machine's CLASSPATH to include its directory.

Testing the Installation of a JDBC-Based Analytics Function

Optional approach: Run it directly on the Aster Database queen

```
beehive@coordinator:~$ java -jar <JDBC-Client-Function>.jar <Command line arguments>
```

Typical approach: Run it from your client machine

```
user@machine:~$ java -classpath <JDBC-Client-Function>.jar:<ncluster-sqlmr-apr>.jar:<JDBC-Driver>.jar <main-class to be invoked with the full package name> <Command line arguments>
```

Command-Line Arguments for Driver-Based Functions

Typically you will pass a set of mandatory arguments and a set of optional arguments.

Mandatory arguments are of two types, generic arguments and function-specific arguments.

The mandatory generic arguments include:

- *domain*: Required argument. Host is the Aster Database queen hostname or IP address. (To specify an IPv6 address, enclose the host parameter in square brackets. For example, "[:: 1]:2406".) The Port is the port number where the Aster Database queen accepts client connections. Default is the Aster Database standard port number (2406). For example: -domain=10.51.23.100:2406
- *database*: Required argument. This is the name of the database where the input table is present. For example: -database=beehive
- *userid*: Required argument. The database user name of the user in Aster Database. For example: -userid=beehive
- *password*: Required argument. The database password of the user in Aster Database. For example: -password=beehive

The rest of the arguments are specific to the function and are listed in the function's documentation.

Using Double-Quotes in Database Object Names

For certain functions, if the name of a database object contains spaces or special characters, you must use double quotes, as shown in the following example.

Create a schema and a table with quoted object names:

```
beehive=> create schema "Schema TxtClssfr";
CREATE SCHEMA
beehive=> create fact table "Schema TxtClssfr"."text_Classifier Case
fact"(id int, "ConTent" text, "[Category] coLumn" text) distribute by
hash(id);
CREATE TABLE
```

The SQL-MR query should be:

```
SELECT *
FROM TextClassifierTrainer(
```

```
on (select 1) PARTITION BY 1
INPUTTABLE('"Schema TxtClssfr"."text_Classifier Case fact"')
textcolumn('ConTent')
categorycolumn('"[Category] coLumn" ')
modelfile('KNN.bin')
classifierType('knn')
classifierParameters('Compress:0.5')
featureSelection('DF:[0.1:]')
DATABASE('beehive')
USERID('beehive')
PASSWORD('beehive')
);
```

The double-quote requirement is limited to several functions (not all the functions):

- textClassifier (textClassifier Trainer, textClassifier, and TextClassifierEvaluator)
- FrequentPaths
- DWT
- IDWT

Creating a Timestamp Column

If there are separate date and time columns in the input table, you can create a timestamp column using this syntax:

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

Best Practices

SQL-MR Capitalization and Quoting Requirements

- When passing arguments to a SQL-MR function, the function treats them as case-sensitive. For example, specifying the column name 'MYCOLUMN' in the SQL-MR function does not find 'mycolumn'.
- When specifying column names in SQL-MR functions, enclose the names in single quotes.
- If you get “Column Not Found” errors, try changing the case of the column name, and rerun the function.

Error Message Delays

In some instances, if there is an error when running a SQL-MR function, the error message might not appear immediately. Instead, it gets displayed after a delay.

CHAPTER 3 Time Series, Path, and Attribution Analysis

- Cumulative Moving Average
- SAX
- Path Generator
- Path Starter
- Path Summarizer
- Sessionization
- Attribution
- FrequentPaths
- DWT
- IDWT
- DWT2d
- IDWT2d
- See also [Pattern Matching with Teradata Aster nPath](#)

Cumulative Moving Average

Summary

The Cumulative Moving Average (CMAVG) function computes the average of a value from the beginning of a series.



Background

In a cumulative moving average, the data are added to the dataset 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. For example, an investor may want to find the average price of all of the stock transactions for a particular stock over time, up until the current time.

The cumulative moving average computes the arithmetic average of all the rows from the beginning of the series:

```
new_smavg = sum(a1 ... aN) / N
```

where N is the number of rows from the beginning of the dataset.

Usage

Permissions

You must grant EXECUTE on the function “CMAVG” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```
SELECT *
FROM CMAVG
(
    ON {table_name|view_name| (query) }
    PARTITION BY partition_column
    ORDER BY order_by_column
    COLUMNS('column_names')
) ;
```

Arguments

COLUMNS

Optional Specifies the column name for which the moving average is to be computed. If this clause is omitted, all the input rows are output as is.

Input Data

Data is assumed to be partitioned such that each partition contains all the rows of an entity. For example, if the cumulative moving average of a particular stock share is required, then all transactions of that share should be part of one partition.

It is assumed that the input rows are provided in the correct time series order. The datatype of the column on which to compute the cumulative moving average must be Double Precision, Integer, BigInteger, or Numeric.

Output

The function outputs the input table, adding a column with a value for each row of the cumulative moving average at the time the row was added to the dataset.

Example

This example computes the cumulative moving average for the price of a stock with ticker symbol APPL.

Example Input Data

Table 3 - 14: Example input table stock_data

symbol	price	ts
APPL	60.33	10/4/2011 4:40
APPL	59.44	10/4/2011 4:41
APPL	59.38	10/4/2011 4:42
APPL	59.38	10/4/2011 4:43
APPL	59.22	10/4/2011 4:44
APPL	59.88	10/4/2011 4:45
APPL	59.55	10/4/2011 4:46
APPL	59.5	10/4/2011 4:47
APPL	58.66	10/4/2011 4:48
APPL	59.05	10/4/2011 4:49
APPL	57.15	10/4/2011 4:50
APPL	57.32	10/4/2011 4:51
APPL	57.65	10/4/2011 4:52
APPL	56.14	10/4/2011 4:53
APPL	55.33	10/4/2011 4:54
APPL	55.86	10/4/2011 4:55
APPL	54.92	10/4/2011 4:56

Table 3 - 14: Example input table stock_data (continued)

symbol	price	ts
APPL	53.74	10/4/2011 4:57
APPL	54.8	10/4/2011 4:58
APPL	54.86	10/4/2011 4:59

Example SQL-MapReduce call

```
SELECT *
FROM CMAVG
(
    ON stock_data
    PARTITION BY symbol
    ORDER BY ts
    COLUMNS('price')
)
ORDER BY ts;
```

Example Output from CMAVG

Table 3 - 15: Example output from CMAVG

symbol	price	ts	price_mavg
APPL	60.33	10/4/2011 4:40	60.33
APPL	59.44	10/4/2011 4:41	59.885
APPL	59.38	10/4/2011 4:42	59.717
APPL	59.38	10/4/2011 4:43	59.6325
APPL	59.22	10/4/2011 4:44	59.55
APPL	59.88	10/4/2011 4:45	59.605
APPL	59.55	10/4/2011 4:46	59.597142857142856
APPL	59.5	10/4/2011 4:47	59.585
APPL	58.66	10/4/2011 4:48	59.48222222222223
APPL	59.05	10/4/2011 4:49	59.439
APPL	57.15	10/4/2011 4:50	59.231
APPL	57.32	10/4/2011 4:51	59.072
APPL	57.65	10/4/2011 4:52	58.962
APPL	56.14	10/4/2011 4:53	58.761
APPL	55.33	10/4/2011 4:54	58.531
APPL	55.86	10/4/2011 4:55	58.364

Table 3 - 15: Example output from CMAVG (continued)

symbol	price	ts	price_mavg
APPL	54.92	10/4/2011 4:56	58.161
APPL	53.74	10/4/2011 4:57	57.916
APPL	54.8	10/4/2011 4:58	57.752
APPL	54.86	10/4/2011 4:59	57.607

Error Messages

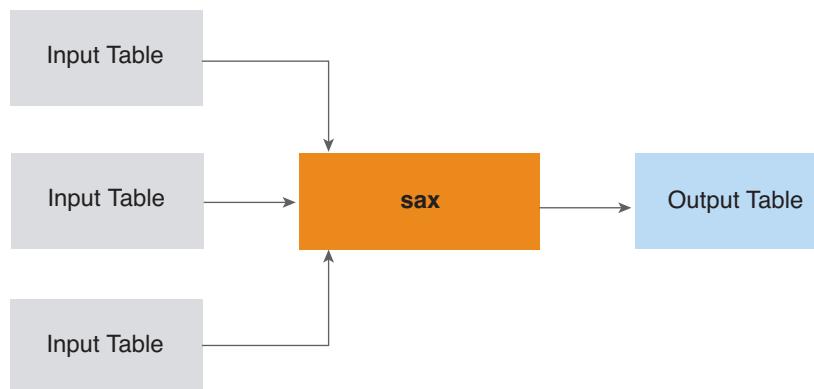
You may see the following error message:

- ERROR: “Moving Average requires datatype of columns to be Double Precision or Integer or BigInteger or Numeric”
REASON: One or more columns specified in the COLUMNS arguments are not of the correct type.

SAX

Summary

Symbolic Aggregate approXimation (SAX) transforms original time series data into symbolic strings. Once this transformation is complete, the data is more suitable for many additional types of manipulation, both because of its smaller size and the relative ease with which patterns can be identified and compared.



Background

A time series is a collection of data observations made sequentially in time. Time series occur in virtually every medical, scientific, entertainment and business domain.

Symbolic Aggregate approXimation uses a simple algorithm, with low computational complexity to create symbolic strings from time series data. Using a symbolic representation

enables additional functions (for example, Teradata Aster nPath) to easily operate on the data.

The data can also be manipulated using common algorithms such as hashing or regular expression pattern matching. In classic data mining tasks such as classification, clustering, and indexing, SAX is accepted as being as good as some well-known, but storage-intensive methods like Discrete Wavelet Transform (DWT) and Discrete Fourier Transform (DFT).

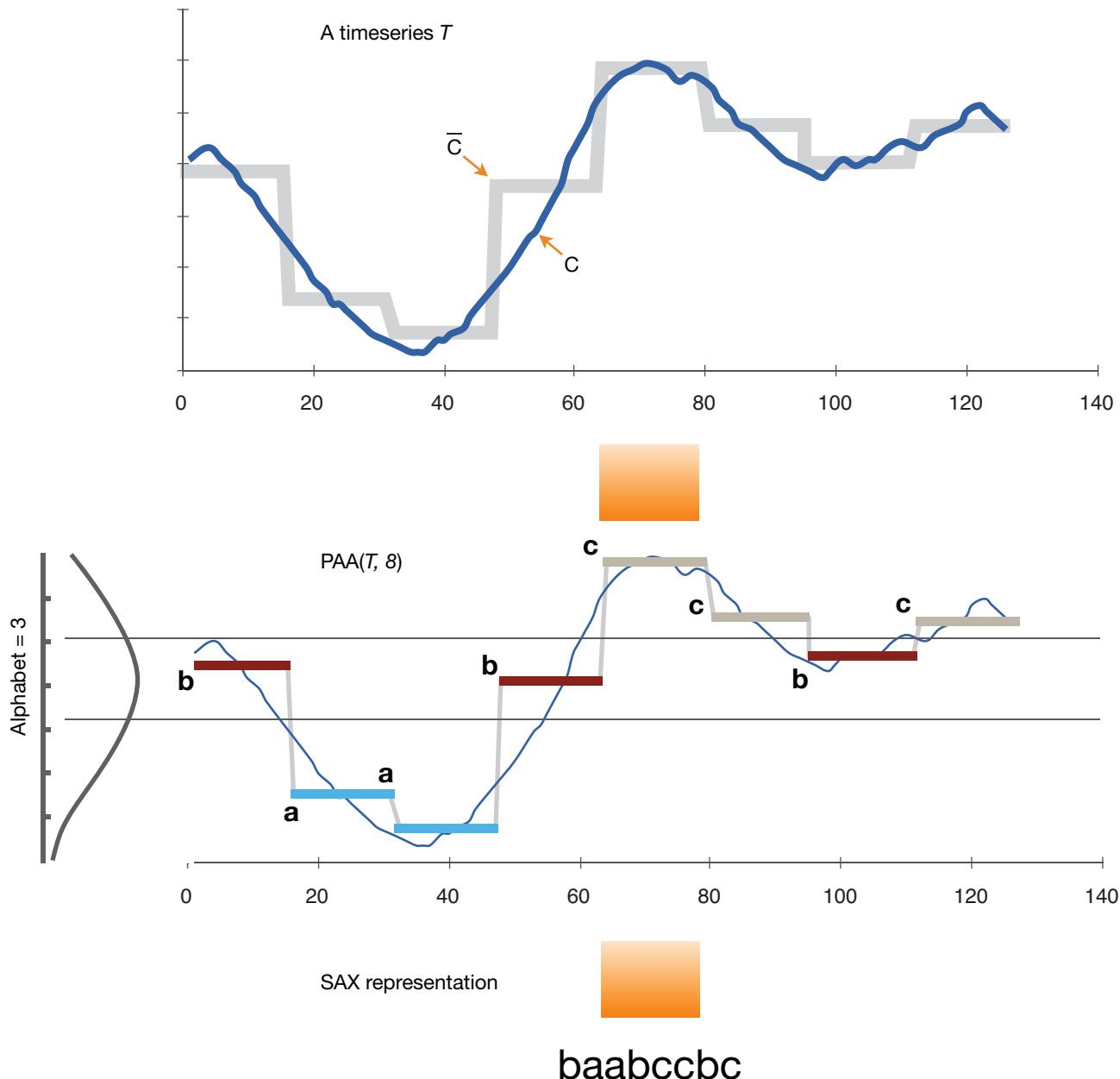
SAX transforms a time series X of length n into the string of arbitrary length w , where $w < n$, using an alphabet A of size $a > 2$.

The SAX algorithm consist of two steps:

- 1 SAX transforms the original time series data into a PAA (Piecewise Aggregate Approximation) representation. This effectively splits the time series data into intervals, and then assigns each interval to one of a limited set of alphabetical symbols (letters) based on the data being examined. The set of symbols used is based on dividing all observed data into “chunks” or thresholds, using the normal distribution curve. Each of these values, or “chunks” is represented by a symbol (a letter). This is a simple way to reduce the dimensionality of the data.
- 2 The PAA is then converted into a string consisting of letters that represents the patterns occurring in the data over time.

The symbols created by SAX correspond to the time series features with equal probability, allowing them to be compared and used for further manipulation with reliable accuracy. The time series that are normalized using the zero mean and unit of energy follow the Normal distribution law. By using Gaussian distribution properties, it's easy for SAX to pick equal-sized areas under the Normal curve using lookup tables for the cut lines coordinates, slicing the under-the-Gaussian-curve area. The x coordinates of these lines are called “breakpoints” in the SAX algorithm context.

This diagram depicts the complete process of saxification:



Usage

Permissions

You must grant EXECUTE on the function “SAX” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (Partition function version 1.1)

```
SELECT * FROM SAX (
    ON {table_name | view_name | (query)}
    PARTITION BY partition_column
    ORDER BY ordering_columns
    VALUE_COLUMN_NAMES('value_column1', 'value_column2', ...)
    [PAA_SIZE('paa_size_value_column1', 'paa_size_value_column2', ...)]
    [MAX_PAA_SIZE('max_paa_value')]
    [ALPHABET_SIZE('alphabet_size_value_column1',
    'alphabet_size_value_column2', ...)]
    [ZNORM('znorm_value_column1', 'znorm_value_column2', ...)]
    [MEAN_VAL('mean_value_column1', 'mean_value_column2', ...)]
    [STDEV_VAL('stdev_value_column1', 'stdev_value_column2', ...)]
    [ACCUMULATE('accumulate_column1', 'accumulate_column2', ...)])
    [TIME_COLUMN_NAME('time_column1', 'time_column2', ...)])
);
```

Syntax (Multiple Input function version 1.1)

```
SELECT * FROM SAX(
    ON {table_name | view_name | (query)} as input
    PARTITION BY partition_columns
    ORDER BY ordering_columns
    ON <(SELECT partition_column1, partition_column2, ... mean_value1 AS
    value_column1, mean_value2 AS value_column2, ... FROM Statistics_Table)>
    as meanstats
    PARTITION BY partition_columns
    ON <(SELECT partition_column1, partition_column2, ... stdev_value1 AS
    value_column1, stdev_value2 AS value_column2, ... FROM
    Statistics_Table)> AS stdevstats
    PARTITION BY partition_columns
    VALUE_COLUMN_NAMES('value_column1', 'value_column2', ...)
    [PAA_INTERVAL('paa_interval_value_column1',
    'paa_interval_value_column2', ...)]
    [ALPHABET_SIZE('alphabet_size_value_column1',
    'alphabet_size_value_column2', ...)]
    [ZNORM('znorm_value_column1', 'znorm_value_column2', ...)]
    [MEAN_VAL('mean_value_column1', 'mean_value_column2', ...)]
    [STDEV_VAL('stdev_value_column1', 'stdev_value_column2', ...)]
    [ACCUMULATE('accumulate_column1', 'accumulate_column2', ...)])
    [TIME_COLUMN_NAME('time_column1', 'time_column2', ...)])
);
```

Arguments

<code>VALUE_COLUMN_NAMES</code>	Required	The column names from the input table that will contain the values of the time series. These columns can be of type int, long, short, double, float, or real.
<code>PAA_SIZE</code>	Required when SAX is used as a partition function; otherwise optional	The PAA size for the time series. PAA size is the number of characters required in the SAX code. Specify either only one value (when the PAA size is unique for all the value columns) or multiple values equal to the number of value columns (when the PAA size is not unique for all the value columns). If the PAA size is not specified, it is set to the default value of 5. The maximum PAA size allowed is defined by the <code>MAX_PAA_SIZE</code> clause.

<i>MAX_PAA_SIZE</i>	Optional	The maximum PAA size to be allowed using this argument clause. This clause can be used only in conjunction with <i>PAA_SIZE</i> . Default is 10000.
<i>PAA_INTERVAL</i>	Required when SAX is used as a multiple-input function; otherwise optional	The PAA interval for the time series. The length of the SAX code is equal to ceil (length of the time series/PAA interval). Specify either only one value (when the PAA interval is unique for all the value columns) or multiple values equal to the number of value columns (when PAA interval is not unique for all the value columns). If the PAA interval is not specified, it is set to the default value, which is equal to 3.
<i>ALPHABET_SIZE</i>	Required	The alphabet size for the time series. The alphabet size determines the number of different characters that can be used as symbols in a SAX representation. Specify either only one value (when the alphabet size is unique for all the value columns) or multiple values equal to the number of value columns (when the alphabet size is not unique for all the value columns). If the alphabet size is not specified, it is set to the default value of 4.
<i>ZNORM</i>	Optional	Determines whether a time series is z-normalized before converting it into a PAA representation. Specify either one value (when the z-normalization option is unique for all the value columns) or multiple values equal to the number of value columns (when the z-normalization option is not unique for all the value columns). Default is 'true'.
<i>MEAN_VAL</i>	Required when ZNORM is false; otherwise optional	The mean value of all the time series of a value column. Specify either only one value (when the mean is unique for all the value columns) or multiple values equal to the number of value columns (when the mean is not unique for all the value columns).
<i>STDEV_VAL</i>	Required when ZNORM is false; otherwise optional	The standard deviation of all the time series of a value column. Specify either only one value (when the standard deviation is unique for all the value columns) or multiple values equal to the number of value columns (when the standard deviation is not unique for all the value columns).
<i>ACCUMULATE</i>	Required	All the columns from input table that should be shown in output.
<i>TIME_COLUMN_NAME</i>	Optional	List of time column names used in the ORDER BY clause if you want to output an associated BEGIN/END value.

Input

The following assumptions apply to the input data:

- All the input time series are uniformly sampled.
- Maximum Alphabet size is 20.
- If a NaN value occurs in the PAA representation of a time series a '_' will be returned in the SAX code.

The following rules apply to the input data for the Multiple-Input function:

The multiple-input function requires three partitioned inputs as specified in the three ON clauses. All the input tables need to be partitioned using the same set of partition keys.

- The multiple-input function is used when the time series is too large to fit in main memory.
- Specify the PAA-interval, so that the PAA-interval size data is read at one time and the SAX code is generated for that part of a time series.
- The SAX code for each PAA interval is combined to get the SAX code of the entire time series.
- The mean and standard deviation of each time series are needed to pass as input for z-normalization.
- Use an SQL query to get the mean and standard deviation of each time series for each value column and store all the returned values in a columnar table.
- Use the alias “input” to refer the table that contains all the time series data.
- Use the alias “meanstats” to refer the table that contains the mean values of each time series for each value column.
- Use the alias “stdevstats” to refer the table that contains the standard deviations of each time series for each value column.

Output

A row is output for each time series of the input table. The output table contains all the columns specified in the VALUE_COLUMN_NAMES clause with the generated SAX code. The output also contains all the columns specified in the ACCUMULATE argument.

Example

Example 1: Partition Function Input Data

Table 3 - 16: Example input data table sax1

id	test_time	value1	value2	quality
1	1	2.02	0.5	good
1	2	2.33	1.29	
1	3	2.99	2.58	
1	4	6.85	3.83	
1	5	9.2	3.25	
1	6	8.8	4.25	
1	7	7.5	3.83	
1	8	6	5.63	
1	9	5.85	6.44	

Table 3 - 16: Example input data table sax1 (continued)

id	test_time	value1	value2	quality
1	10	3.85	6.25	
1	11	4.85	8.75	
1	12	3.85	8.83	
1	13	2.22	3.25	
1	14	1.45	0.75	
1	15	1.34	0.72	

Example 1: Partition Function SQL-MapReduce call

```
SELECT * from SAX (
    ON (SELECT id, test_time, value1, value2, value1 AS value3,
        value2 AS value4 FROM sax1)
    PARTITION BY id
    ORDER BY test_time
    VALUE_COLUMN_NAMES('value1','value2','value3','value4')
    PAA_SIZE('5')
    ALPHABET_SIZE('4')
    ZNORM('true','true','false','false')
    MEAN_VAL('4.60667','4.01','4.60667','4.01')
    STDEV_VAL('2.64032','2.72213','2.64032','2.72213')
    ACCUMULATE('id')
) ;
```

Example 1: Partition Function Output

Table 3 - 17: Example 1: Partition Function output from CMAVG

id	sax_code_value1	sax_code_value2	sax_code_value3	sax_code_value4
1	addba	abcd	addba	abcd

Example 2: Multiple Input Function Input Data

In addition to the table sax1, shown as input to Example 1, Example 2 uses the statssax1 table.

Table 3 - 18: Example 2: Multiple Input Function input table statssax1

id	meanvalue1	stdevvalue1	meanvalue2	stdevvalue2	quality
1	4.6067	2.64032	4.01	2.7221	good

Example 2: Multiple Input Function SQL-MapReduce call

```
SELECT * FROM SAX (
    ON (SELECT id, test_time, value1, value2, value1 AS value3, value2 AS
        value4 FROM sax1) AS INPUT
    PARTITION BY id
    ORDER BY test_time
    ON (SELECT id, meanvalue1 AS value1, meanvalue2 AS value2, meanvalue1
        AS value3, meanvalue2 AS value4 FROM statssax1) AS meanstats
    PARTITION BY id
    ON (SELECT id, meanvalue1 AS value1, meanvalue2 AS value2, meanvalue1
        AS value3, meanvalue2 AS value4 FROM statssax1) AS stdevstats
    PARTITION BY id
    VALUE_COLUMN_NAMES('value1','value2','value3','value4')
    PAA_INTERVAL('3')
    ALPHABET_SIZE('4')
    ZNORM('true','true','false','false')
    MEAN_VAL('4.60667','4.01','4.60667','4.01')
    STDEV_VAL('2.64032','2.72213','2.64032','2.72213')
    ACCUMULATE('id')
);
```

Example 2: Multiple Input Function Output

Table 3 - 19: Example 2: Multiple Input Function output from SAX

id	sax_code_value1	sax_code_value2	sax_code_value3	sax_code_value4
1	bdcbb	bbcdcb	addba	abcd

Example 3: SQL-MapReduce call (TIME_COLUMN_NAME)

```
select * from sax (
    ON (select id, test_time, value1, value2, value1 as value3, value2 as
        value4 from sax1)
    partition by id
    order by test_time
    VALUE_COLUMN_NAMES('value1','value2','value3','value4')
    PAA_SIZE('5')
    ALPHABET_SIZE('4')
    ZNORM('true','true','false','false')
    MEAN_VAL('4.60667','4.01','4.60667','4.01')
    STDEV_VAL('2.64032','2.72213','2.64032','2.72213')
    ACCUMULATE('id')
    TIME_COLUMN_NAME('test_time')
);
```

Example 3: Output

Table 3 - 20: Example 3: Output

id	begin_test	end_test	sax_code_value1	sax_code_value2	sax_code_value3	sax_code_value4
1	1	15	addba	abcd	addba	abcd

Path Generator

Summary

This function takes as input a set of *paths* where each path is a route (series of pageviews) taken by a user in a single session from the start of the session until its end. For each path, Path Generator generates the correctly formatted *sequence* and all possible *sub-sequences* for further analysis by the Path Summarizer function. The first element in the path is the first page a user could visit. The last element of a path is the last page visited by the user.



Together, the Path Generator, Path Summarizer, and Path Starter functions are used to perform clickstream analysis of common sequences of users' pageviews on a website. The roles of the three functions are:

- Path Generator generates all the possible paths (sequences of pageviews on a website);
- Path Summarizer counts the number of times various paths were traveled and measures the depth in pageviews of each path; and
- Path Starter generates all the child paths for a particular parent path and sums up the count of times each child path was traveled.

In the discussion below, we will use the terms:

- Path: An ordered, start-to-finish series of actions (for example, pageviews) for which you wish to generate sequences and sub-sequences. You will run Path Generator on the set of all observed paths users have traveled while navigating your website.
- Sequence: The sequence is the path prefixed with a carat (^) to indicate the start of the path. For example, if a user visited page a, page b, and page c in that order, we would say that his session had the sequence, ^, a, b, c.
- Sub-sequence: 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,c generates three sub-sequences: ^, a; ^, a, b; and ^, a, b, c.

Background

This tool is useful for performing clickstream analysis of website traffic. These functions can also be used for doing other types of sequence/path analysis, such as the analysis required for advertisement attribution and referral attribution.

Usage

Permissions

You must grant EXECUTE on the function “path_generator” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.1)

```
SELECT *
  FROM path_generator
  (
    ON {table_name | view_name | (query)}
    SEQ('sequence_column')
    [DELIMITER('delimiter_character')]
  );
```

Arguments

SEQ	Required	Name of the column in the input relation that contains the paths to be analyzed. The SEQ column must be of type <i>varchar</i> . Each path string is a delimited list of alphanumeric symbols that represents an ordered sequence of pageviews (or actions). Typically each symbol is a code that represents a unique pageview.
DELIMITER	Optional	Specifies the single-character delimiter you used in the path string (default is “,”).

Input Data

You specify your *path* column in the SEQ parameter. The data source can be rows from a table or the result of a Teradata Aster nPath query. In the query that generates the input data, you must GROUP BY your path column so that there is one row for each unique path traveled on your website, with a count of times that path was traveled.

Output

The function emits a row for each sub-sequence it generates. Each output row contains:

- The sub-sequence (called the “prefix”)
- The formatted sequence with a carat (“^”) character added as the first element in the sequence; the carat indicates the start of the sequence. For example, a sequence in which a user viewed pages a, b, c, and then d would be expressed as the sequence ^, a, b, c, d.
- The path and all other columns of the input row that generated this sub-sequence.

Example

The input table *user_flows* contains the columns *user_id*, *path*, and *cnt*. Path Generator operates only on the *path* data, and outputs the other columns' data untouched.

Example Input Data

Table 3 - 21: Example input table user_flows

user_id	path	cnt
1	a,b,c,d	1
2	a,b	2
3	b,e,g	5
4	a	7
5	a,e	5

Example SQL-MapReduce call

```
SELECT *
  FROM PATH_GENERATOR
  (
    ON user_flows
    SEQ('path')
    DELIMITER( ', ' )
  )
order by user_id;
```

Example Output from Path Generator

As mentioned above, the function generates a row for every sub-sequence it generates. The sub-sequence itself is output in the “prefix” column. All input columns (*user_id*, *path*, and *cnt*, here) are returned in the results, as well.

Table 3 - 22: Example output from Path Generator

user_id	path	cnt	prefix	sequence
1	a,b,c,d	1	^,a	^,a,b,c,d
1	a,b,c,d	1	^,a,b	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c,d	^,a,b,c,d
2	a,b	2	^,a	^,a,b
2	a,b	2	^,a,b	^,a,b
3	b,e,g	5	^,b	^,b,e,g
3	b,e,g	5	^,b,e	^,b,e,g
3	b,e,g	5	^,b,e,g	^,b,e,g
4	a	7	^,a	^,a

Table 3 - 22: Example output from Path Generator (continued)

user_id	path	cnt	prefix	sequence
5	a,e	5	^,a	^,a,e
5	a,e	5	^,a,e	^,a,e

Error Messages

You may see the following error message:

- ERROR: “Please provide the SEQ argument, usage: SEQ(<columnname>)”
REASON: SEQ argument is missing

Path Starter

Summary

The output of Path Summarizer function is the input to this function. This function generates all the children for a particular parent and sums up their count. Note that the input data has to be partitioned by the parent column.



Background

This function is useful for website clickstream analysis and other sequence/path analysis tasks such as advertisement attribution.

Usage

Permissions

You must grant EXECUTE on the function “path_start” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.0)

```

SELECT *
  FROM PATH_START
  (
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    CNT('count-column')
    [DELIMITER(',')]
```

```

PARENT('parent-column')
PARTITIONNAMES('partitionby-col-name' [ , ... ] )
NODE('node-column')
);

```

Arguments

<i>CNT</i>	Required	Name of the column that contains the count values.
<i>DELIMITER</i>	Optional	Single-character delimiter to use (default is ",").
<i>PARENT</i>	Required	Name of the column that contains the path of the parent node. You must partition the input data on this column.
<i>PARTITIONNAMES</i>	Required	This is a comma-delimited list of names to be used in the output as the names of the partition-by columns. Make sure you specify one name in this clause for each column you included in the PARTITION BY clause.
<i>NODE</i>	Required	Name of column that contains the path of the current node.

Assumptions

The user is expected to partition the input data based on the PARENT column (required) and optionally on any additional columns. All the columns used for partitioning will be emitted as-is. The PARTITIONNAMES argument clause is used to give output-column names to all the columns that are being used to partition the input data set. The number of columns in the PARTITION BY clause must be the same as the number of names in the PARTITIONNAMES argument. The NODE argument clause is used to specify the column containing the path to the node.

Input Data

Input column datatype requirements:

- The *node* column and the *parent* column should be of type *varchar*.
- The *cnt* column should be of type *bigint* or *int*.

All other columns are ignored unless they are part of the partition function.

Output

The function emits *node*, *parent*, *children*, *cnt*, *depth*, and all columns that are part of the partition function.

Example

Input table *user_flow_subpaths* contains node, cnt, parent, prefix, depth, and children.

Example Input Data

Table 3 - 23: Example input table: user_flow_subpaths

node	cnt	parent	prefix	depth	children
^,a	15	^	^,a	1	[(^,a,\$),(^,a,b),(^,a,e)]
^,a,\$	7	^,a	^,a	2	
^,a,b	3	^,a	^,a,b	2	[(^,a,b,\$),(^,a,b,c)]
^,a,b,\$	2	^,a,b	^,a,b	3	
^,a,b,c	1	^,a,b	^,a,b,c	3	[(^,a,b,c,d)]
^,a,b,c,d	1	^,a,b,c	^,a,b,c,d	4	[(^,a,b,c,d,\$)]
^,a,b,c,d,\$	1	^,a,b,c,d	^,a,b,c,d	5	
^,a,e	5	^,a	^,a,e	2	[(^,a,e,\$)]
^,a,e,\$	5	^,a,e	^,a,e	3	
^,b	5	^	^,b	1	[(^,b,e)]
^,b,e	5	^,b	^,b,e	2	[(^,b,e,g)]
^,b,e,g	5	^,b,e	^,b,e,g	3	[(^,b,e,g,\$)]
^,b,e,g,\$	5	^,b,e,g	^,b,e,g	4	

Example SQL-MapReduce call

```
SELECT *
  FROM PATH_START
  (
    ON user_flow_subpaths
    PARTITION BY (parent)
    CNT('cnt')
    DELIMITER(',')
    PARENT('parent')
    PARTITIONNAMES('partitioned')
    NODE('node')
  )
  order by node;
```

Example Output from Path Start

Table 3 - 24: Example PATH_START output table

node	parent	children	cnt	depth	partitioned
^		[(^,a),(^,b)]	20	0	^
^,a	^	[(^,a,\$),(^,a,b),(^,a,e)]	15	1	^,a
^,a,b	^,a	[(^,a,b,\$),(^,a,b,c)]	3	2	^,a,b
^,a,b,c	^,a,b	[(^,a,b,c,d)]	1	3	^,a,b,c
^,a,b,c,d	^,a,b,c	[(^,a,b,c,d,\$)]	1	4	^,a,b,c,d
^,a,e	^,a	[(^,a,e,\$)]	5	2	^,a,e

Table 3 - 24: Example PATH_START output table (continued)

node	parent	children	cnt	depth	partitioned
^,b	^	[(^,b,e)]	5	1	^, b
^,b,e	^,b	[(^,b,e,g)]	5	2	^,b,e
^,b,e,g	^,b,e	[(^,b,e,g,\$)]	5	3	^,b,e,g

Path Summarizer

Summary

The output of the Path Generator function is the input to this function. This function is used to sum counts on *nodes*. A *node* can either be a *plain sub-sequence* or an *exit sub-sequence*. An exit sub-sequence is one in which the sequence and the sub-sequence are the same. Exit sub-sequences are denoted by appending a dollar sign ('\$') to the end of the sequence.



Background

This function is useful for website clickstream analysis and other sequence/path analysis tasks such as advertisement attribution.

Usage

Permissions

You must grant EXECUTE on the function “path_summarizer” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.0)

```

SELECT *
  FROM PATH_SUMMARIZER
  (
    ON {table_name|view_name|(query)}
    PARTITION BY expression [, ...]
    CNT('count_column')
    DELIMITER(',')
    SEQ('sequence-column')
    PARTITIONNAMES('partitionby-col-name' [ , ... ] )
    HASH('true|false')
    PREFIX('prefix-column')
  );

```

Arguments

<i>CNT</i>	Required	Name of the column containing the count values. If an input row has no <i>CNT</i> value, then the row is assumed to have a count of 1.
<i>DELIMITER</i>	Optional	Single-character delimiter to use (default is ",").
<i>SEQ</i>	Required	Name of the column containing the path of the current node.
<i>PARTITIONNAMES</i>	Required	Names for the columns specified in the PARTITION BY clause. The number of names specified in this argument must match the number of columns in the PARTITION BY clause.
<i>HASH</i>	Optional	Boolean that specifies whether the hash code of the NODE column should be included in the output. (Default is "false").
<i>PREFIX</i>	Required	Name of the column containing the prefix of a given node.

Assumptions

The user is expected to partition the input data on the *PREFIX* column (required) and optionally on additional columns. All the columns used for partitioning will be emitted as-is. The *PARTITIONNAMES* argument clause is used to name all the columns being used to partition the input data set. The number of columns in the PARTITION BY clause must be same as the number of names in the *PARTITIONNAMES* argument. Use the *SEQ* argument clause to specify the column containing the path to the node.

Input Data

All other columns are ignored unless part of the partition function. Observe the following datatype requirements for input columns to Path Summarizer:

- The *PREFIX* and *SEQ* columns should be of type *varchar*.
- The *CNT* column should be of type *bigint* or *int*.

Output

The function emits the node, parent to the node, children, cnt (the sum of input counts), depth, and columns part of the partition function. The parent of the node is the route traversed by the users before visiting this node. Children of the node are the set of routes traversed by the users after visiting this node. Depth is the number of elements visited before entering this node.

Example

Input table `output_of_path_generator` contains user_id, path, prefix, sequence, and cnt.

Example Input Data

Table 3 - 25: Example input table: output_of_path_generator

user_id	path	cnt	prefix	sequence
1	a,b,c,d	1	^,a	^,a,b,c,d
1	a,b,c,d	1	^,a,b	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c	^,a,b,c,d
1	a,b,c,d	1	^,a,b,c,d	^,a,b,c,d
2	a,b	2	^,a	^,a,b
2	a,b	2	^,a,b	^,a,b
3	b,e,g	5	^,b	^,b,e,g
3	b,e,g	5	^,b,e	^,b,e,g
3	b,e,g	5	^,b,e,g	^,b,e,g
4	a	7	^,a	^,a
5	a,e	5	^,a	^,a,e
5	a,e	5	^,a,e	^,a,e

Example SQL-MapReduce call

```

SELECT *
FROM PATH_SUMMARIZER
(
    ON (
        SELECT *
        FROM PATH_GENERATOR
        (
            ON user_flows
            SEQ('path')
            DELIMITER(',')
        )
    )
    PARTITION BY prefix
    SEQ('sequence')
    PREFIX('prefix')
    PARTITIONNAMES('prefix')
    DELIMITER(',')
    CNT('cnt')
    HASH('false')
) order by node;

```

Example Output from Path Summarizer

Table 3 - 26: Example output table from Path Summarizer

node	parent	children	cnt	depth	prefix
^,a	^	[(^,a,\$), (^,a,b), (^,a,e)]	15	1	^,a
^,a,\$	^,a		7	2	^,a
^,a,b	^,a	[(^,a,b,\$), (^,a,b,c)]	3	2	^,a,b
^,a,b,\$	^,a,b		2	3	^,a,b
^,a,b,c	^,a,b	[(^,a,b,c,d)]	1	3	^,a,b,c
^,a,b,c,d	^,a,b,c	[(^,a,b,c,d,\$)]	1	4	^,a,b,c,d
^,a,b,c,d,\$	^,a,b,c,d		1	5	^,a,b,c,d
^,a,e	^,a	[(^,a,e,\$)]	5	2	^,a,e
^,a,e,\$	^,a,e		5	3	^,a,e
^,b	^	[(^,b,e)]	5	1	^,b
^,b,e	^,b	[(^,b,e,g)]	5	2	^,b,e
^,b,e,g	^,b,e	[(^,b,e,g,\$)]	5	3	^,b,e,g
^,b,e,g,\$	^,b,e,g		5	4	^,b,e,g

Error Messages

You may see one or more of the following error messages when you attempt to use this function:

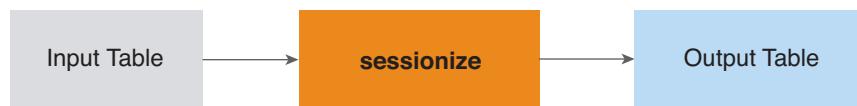
- ERROR: Please provide the SEQ argument, usage: SEQ(<columnname>)
REASON: SEQ argument is missing
- ERROR: Please specify the column containing the prefix
REASON: PREFIX argument is missing
- ERROR: Please specify the column containing the cnt
REASON: CNT argument is missing
- ERROR: Please specify the names for the partition columns
REASON: PARTITIONNAMES argument is missing

Sessionization

Background

Sessionization is the process of mapping each click in a clickstream to a unique session identifier. We define a session as a sequence of clicks by a particular user where no more than n seconds pass between successive clicks (that is, if we don't see a click from a user for n seconds, we start a new session).

Sessionization can be easily done with the Sessionize SQL-MapReduce function. Sample code is included with the Aster SQL-MapReduce Java API. This sessionize SQL-MapReduce function can also be used to detect web crawler (“bot”) activity. If the time between successive clicks is less than the user-specified threshold, bot activity will be flagged.



Usage

Permissions

You must grant EXECUTE on the function “sessionize” to the database user who will run the function. Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.2)

```

SELECT *
  FROM SESSIONIZE(
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    ORDER BY order_by_columns
    TIMECOLUMN ('timestamp_column')
    TIMEOUT (session_timeout_value)
    [RAPIDFIRE (min_human_click_lag)]
    [EMITNULL]
  );
  
```

Arguments

<i>TIMECOLUMN</i>	Required	Specifies the column name containing the time information. The specified column can be of type TIME, TIMESTAMP, INT, BIGINT, or SMALLINT. If the column is of type INT, BIGINT, or SMALLINT, it is assumed to contain timestamp values in milliseconds.
<i>TIMEOUT</i>	Required	Specifies the maximum number of seconds a user can wait between one pageview and the next, before it the new pageview is considered to be part of a new session. This value can have the datatype REAL.
<i>RAPIDFIRE</i>	Optional	Specifies minimum number of seconds that must elapse between clicks in order for this session to be considered a real (human) session. If the time between clicks is less than the <i>min_human_click_lag</i> , SESSIONIZE considers the session to be a bot session and ignores it. RAPIDFIRE must be less than TIMEOUT. The data type of this value is REAL.

<i>EMITNULL</i>	Optional If true, emits the row with null values for sessionid and rapidfire even if the TIMECOLUMN has null value. If false, rows with null values for TIMECOLUMN would not be emitted. By default EMITNULL is false.
-----------------	--

Assumptions

Data is assumed to be partitioned such that each partition contains all the rows of an entity. Since the function output includes a column called 'sessionid', the input may not include a column of the same name.

Creating a Timestamp Column

If there are separate date and time columns in the input table, you can create a timestamp column using this syntax:

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

Example

Example Input Data

Table 3 - 27: Example input table: sessionize_table

partition_id	clicktime	userid	productname	pagetype	referrer	productprice
1	10/10/2008 7:00	333		home	www.yahoo.com	
1	10/10/2008 7:00	333	ipod	checkout	www.yahoo.com	200.2
1	10/10/2008 7:01	333	bose	checkout		340
1	10/12/2008 15:34	333		home	www.google.com	
1	8/12/2007 15:35	67403		home	www.google.com	
1	8/12/2007 13:18	67403		home	www.google.com	
1	8/12/2007 13:18	67403		home		
1	8/12/2007 13:18	67403		home		
1	8/12/2007 13:18	67403	iphone	checkout		650
1	8/12/2007 13:19	67403	bose	checkout		750
1	8/12/2007 20:00	80000		home	www.godaddy.com	
1	8/12/2007 20:02	80000	bose	checkout		340
1	8/12/2007 20:02	80000	itrip	checkout		450
1	8/12/2007 20:03	80000	iphone	checkout		650

Example SQL-MapReduce call

```
SELECT *
  FROM SESSIONIZE
  (
    ON sessionize_table
    PARTITION BY partition_id
```

```

        ORDER BY clicktime
        TIMECOLUMN('clicktime')
        TIMEOUT('60')
        RAPIDFIRE('0.2')
    )
    ORDER BY partition_id, clicktime;

```

Example Output from Sessionize

Table 3 - 28: Example output table

partition_id	click time	userid	product name	pagetype	referrer	product price	session id	rapid fire
1	8/12/2007 13:18	67403		home	www.google.com	0	f	
1	8/12/2007 13:18	67403		home		0	f	
1	8/12/2007 13:18	67403		home		0	f	
1	8/12/2007 13:18	67403	iphone	checkout		650	0	f
1	8/12/2007 13:19	67403	bose	checkout		750	0	f
1	8/12/2007 15:35	67403		home	www.google.com	1	f	
1	8/12/2007 20:00	80000		home	www.godaddy.com	2	f	
1	8/12/2007 20:02	80000	bose	checkout		340	3	f
1	8/12/2007 20:02	80000	itrip	checkout		450	3	f
1	8/12/2007 20:03	80000	iphone	checkout		650	3	f
1	10/10/2008 7:00	333		home	www.yahoo.com	4	f	
1	10/10/2008 7:00	333	ipod	checkout	www.yahoo.com	200.2	4	f
1	10/10/2008 7:01	333	bose	checkout		340	4	f
1	10/12/2008 15:34	333		home	www.google.com	5	f	

Output contains all the input columns; in addition it contains sessionid and rapidfire columns.

Error Messages

You may see the following error messages when using this function:

- ERROR: Requires specified timecolumn column (<column_name>) to have any of the following types:integer, smallint, bigint, timestamp, time
REASON: column specified in the TIMECOLUMN argument is not among any of the allowed datatypes.
- ERROR: TIMEOUT should be a real value greater than 0
REASON: TIMEOUT argument is not a real value greater than 0
- ERROR: RAPIDFIRE should be a real value greater than 0
REASON: RAPIDFIRE argument is not a real value greater than 0
- ERROR: rapidfire should be less than the timeout
REASON: RAPIDFIRE argument should be less than the TIMEOUT argument

Attribution

Summary

There are two different ways to call the attribution SQL-MR function, depending on how many inputs you are using. The multiple input and the single input function calls use very different syntax, so they are documented separately:

- [Attribution \(Multiple Input\)](#)
- [Attribution \(Single Input\)](#)

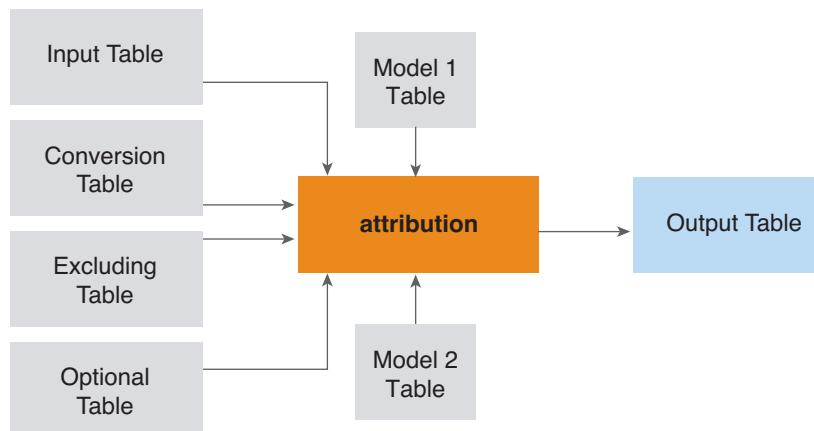
Permissions

You must grant EXECUTE on the function “attribution” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Attribution (Multiple Input)

The multiple-input version of the attribution function lets you specify some parameters in input tables, instead of in argument clauses as in the single-input version of this function. This multiple-input version is especially useful when the number of parameters is large.



Implementation Notes

When running attribution with multiple inputs, all queries should finish running in 2–3 seconds because the input data is between 12 and 15 tuples. If these functions run for a longer period of time before displaying the output, then this is an indication that some of the arguments supplied to the function are incorrect.

Usage

Syntax (Multiple Input version 1.0)

```
SELECT * FROM attribution
(
    ON {input_table | view | query}
    AS input PARTITION BY user_id
    ORDER BY time_stamp

    ON conversion_event_table AS conversion DIMENSION
    ON excluding_event_table AS excluding DIMENSION
    ON optional_event_table AS optional DIMENSION

    ON model1_table AS model1 DIMENSION
    ON model2_table AS model2 DIMENSION

    EVENT_COLUMN_NAME('event_column')
    TIMESTAMP_COLUMN_NAME('timestamp_column')
    WINDOW('rows:K | seconds:K | rows:K&seconds:K2')

) ORDER BY user_id, time_stamp;
```

Simple arguments

EVENT_COLUMN_NAME	Required	Name of the event column.
TIMESTAMP_COLUMN_NAME	Required	Name of the timestamp column. The column type can be integer, smallint, bigint, timestamp, and time.
WINDOW	Required	Maximum window size used in the attribution calculation. There are three window size modes: <ul style="list-style-type: none"> The 'rows:K' mode considers the maximum number of cause events to be attributed, excluding cause events with an event type value specified in the 'excludingEventTypeValue' argument, which means assigning attributions to at most K effective cause events before current impact event. The 'seconds:K' considers the maximum time difference between current impact event and the earliest effective cause event to be attributed. And the mixed mode 'rows:K&seconds:K2' considers both constraints and complies to the stricter one.

Input tables

input_table	Required	Contains the click stream data, which the function uses to compute attributions.
conversion	Required	One-column table containing varchar values of conversion events.
excluding	Required	One-column table containing varchar values of excluding cause events.
optional	Required	One-column table containing varchar values of optional cause events.

model1 and model2 arguments

The first row in the model1 and model2 tables defines the model type. The remaining rows in the tables define the distribution models. This table describes the format of the model1 table.

Table 3 - 29: model1 table format

id (int)	model (varchar)
0	'TYPE'
1	'K EVENT:WEIGHT:MODEL:PARAMETERS'
2	...

This table describes the format of the model2 table.

Table 3 - 30: model2 table format

id (int)	model (varchar)
0	'TYPE'
1	'K EVENT:WEIGHT:MODEL:PARAMETERS'
2	...

TYPE: The values of **TYPE** can be:

'SIMPLE'	Specifies a single distribution model in the next row to be used for all the events. In this case, the format of the distribution model is ' MODEL:PARAMETERS '.
'EVENT_REGULAR'	Specifies a list of regular event distribution models in the following rows. The format of these models is ' EVENT:WEIGHT:MODEL:PARAMETERS '. All the weights in the table must add up to 1.
'EVENT_OPTIONAL'	Specifies a list of optional event distribution models in the following rows. The format of these models is ' EVENT:WEIGHT:MODEL:PARAMETERS '. All the weights in the table must add up to 1. The value of EVENT should be one of the values defined in the optional input table. Note that when specifying ' OPTIONAL_EVENT_TYPE_VALUE ' and ' EVENT_REGULAR ', ' EVENT_OPTIONAL ' is required.
'SEGMENT_ROWS'	Specifies a list of window slices by rows in the following rows, starting from the most recent event to the oldest. The format of these models is ' K:EVENT:WEIGHT:MODEL:PARAMETERS '. All the weights in the table must add up to 1. All of the K_i 's must add up to K as specified in the ' rows:K ' mode defined by the WINDOW argument. Similarly, all of the K values must add up to K , as specified in the ' rows:K ' mode.
'SEGMENT_SECONDS'	Specifies a list of window slices by seconds in the following rows, starting from the most recent event to the oldest. The format of these models is ' K:EVENT:WEIGHT:MODEL:PARAMETERS '. All the weights in the table must add up to 1. All of the K_i 's must add up to K as specified in the ' seconds:K ' mode defined by the WINDOW argument. Similarly, all of the K values must add up to K , as specified in the ' seconds:K ' mode.

The allowed MODEL1/MODEL2 combinations are:

- MODEL1('SIMPLE')
- MODEL1('EVENT_REGULAR')
- MODEL1('EVENT_REGULAR'), MODEL2('EVENT_OPTIONAL')

Note that when specifying 'OPTIONAL_EVENT_TYPE_VALUE' and 'EVENT_REGULAR', 'EVENT_OPTIONAL' is required.
- MODEL1('SEGMENT_ROWS')
- MODEL1('SEGMENT_SECONDS')
- MODEL1('SEGMENT_ROWS'), MODEL2('SEGMENT_SECONDS')

You must specify 'rows:K&seconds:K' in the WINDOW argument.

MODEL: The model used by this function to calculate the attribution. Currently supported models are:

- 'LAST_CLICK'
- 'UNIFORM'
- 'WEIGHTED'
- 'EXPONENTIAL'
- 'FIRST_CLICK'.

This argument is case sensitive.

PARAMETERS:

- If the distribution model is 'LAST_CLICK', 'UNIFORM', or 'FIRST_CLICK', PARAMETERS must be 'NA'. If not, you get an error.
- If the distribution model is 'WEIGHTED':
 - If the mode is 'rows:K', you must specify a list of weights whose size is equal to K (which is specified in the WINDOW argument), or equal to K_i (which is specified in ' $K_i:MODEL_i:WEIGHT_i$ ').
 - If the mode is 'seconds:K' or if it is an event model, you can specify as many weights as you want because they will be dynamically re-normalized. Each weight should be in range [0,1], and all the weights must add up to 1. Specify the weights from newest (left) to oldest (right) attributed cause event.

To better understand the weight semantics, consider this sequence:

“impression1,impression2,click1,impression3,click2,click3”

The window is row-based for the three preceding rows and the user-specified weights are “0.5,0.3,0.2.” For click1, impression1 has an attribution of 0.375 (0.3 normalized by $0.3+0.5$) and impression2 has an attribution of 0.625 (0.5 normalized by $0.3+0.5$). For click2, there is only 1 qualifying cause event: impression3 has an attribution 1.0 (0.5 normalized by 0.5).

- If the distribution model is 'EXPONENTIAL', you need to specify a single parameter 'alpha' in range (0,1). Consider this power series: $\omega_i = (1-\alpha) \times \alpha^i$, $0 < \alpha < 1$, $i = 0, 1, \dots$ which adds up to 1. These exponential weights can be considered relative weights when the actual rows being looked back are not infinite.

Example (Multiple Input)

Example Input Tables

This table displays the rows of the input table *attribution_sample_table*.

Table 3 - 31: Example input table: attribution_sample_table

user_id	event	time_stamp
1	impression	2001-09-27 23:00:01
1	impression	2001-09-27 23:00:03
1	impression	2001-09-27 23:00:05
1	impression	2001-09-27 23:00:07
1	impression	2001-09-27 23:00:09
1	impression	2001-09-27 23:00:11
1	impression	2001-09-27 23:00:13
1	email	2001-09-27 23:00:15
1	impression	2001-09-27 23:00:17
1	impression	2001-09-27 23:00:19
1	click1	2001-09-27 23:00:20
1	optional1	2001-09-27 23:00:21
1	optional2	2001-09-27 23:00:22
1	click2	2001-09-27 23:00:23
2	impression	2001-09-27 23:00:29
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
2	impression	2001-09-27 23:00:43
2	impression	2001-09-27 23:00:47
2	optional	2001-09-27 23:00:49
2	impression	2001-09-27 23:00:51
2	impression	2001-09-27 23:00:53
2	impression	2001-09-27 23:00:55
2	click1	2001-09-27 23:00:59

This table displays the rows of the input table *conversion_event_table*.

Table 3 - 32: Example input table: conversion_event_table

conversion_events
click1
click2

This table displays the rows of the input table *excluding_event_table*.

Table 3 - 33: Example input table: excluding_event_table

excluding_events
email

This table displays the rows of the input table optional_event_table.

Table 3 - 34: Example input table: model1_table

optional_events
optional
optional1
optional2

This table displays the rows of the input table model1_table.

Table 3 - 35: Example input table: model1_table

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

This table displays the rows of the input table model2_table.

Table 3 - 36: Example input table: model2_table

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

Example SQL-MR function call

```
SELECT * FROM attribution
(
    ON attribution_sample_table AS input PARTITION BY user_id
        ORDER BY time_stamp
    ON conversion_event_table AS conversion DIMENSION
    ON excluding_event_table AS excluding DIMENSION
    ON optional_event_table AS optional DIMENSION
    ON model1_table AS model1 DIMENSION
    ON model2_table AS model2 DIMENSION

    EVENT_COLUMN_NAME('event')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
)
ORDER BY user_id, time_stamp;
```

Output table

This table displays the rows of the table generated by the attribution function in this example.

Table 3 - 37: Example output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0.2	-19
1	impression	2001-09-27 23:00:03	0	
1	impression	2001-09-27 23:00:05	0.03	-15
1	impression	2001-09-27 23:00:07	0.06	-13
1	impression	2001-09-27 23:00:09	0.09	-11
1	impression	2001-09-27 23:00:11	0.12	-9
1	impression	2001-09-27 23:00:13	0.0714286	-7
1	impression	2001-09-27 23:00:17	0.142857	-3
1	impression	2001-09-27 23:00:19	0.285714	-1
1	click1	2001-09-27 23:00:20		
1	optional1	2001-09-27 23:00:21	0.333333	-2
1	optional2	2001-09-27 23:00:22	0.666667	-1
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:36	0	
2	impression	2001-09-27 23:00:38	0	
2	impression	2001-09-27 23:00:43	0.2	-16

Table 3 - 37: Example output table (continued)

user_id	event	time_stamp	attribution	time_to_conversion
2	impression	2001-09-27 23:00:47	0	
2	impression	2001-09-27 23:00:51	0.3	-8
2	impression	2001-09-27 23:00:53	0.25	-6
2	impression	2001-09-27 23:00:55	0.25	-4

Attribution (Single Input)

This section describes the single input version of the Attribution function.



Usage

Syntax (*Single Input function version 1.0*)

```

SELECT * FROM attribution
(
    ON {input_table | view | query}
    PARTITION BY expression [, ...]
    ORDER BY order_by_columns
    EVENT_COLUMN_NAME('event_column')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2', ...)
    [EXCLUDING_EVENT_TYPE_VALUE('email')]
    [OPTIONAL_EVENT_TYPE_VALUE('optional1', 'optional2')]
    TIMESTAMP_COLUMN_NAME('timestamp_column')
    WINDOW('rows:K | seconds:K | rows:K&seconds:K')
    MODEL1('TYPE', 'K|EVENT:WEIGHT:MODEL:PARAMETERS', ...)
    [MODEL2('TYPE', 'K|EVENT:WEIGHT:MODEL:PARAMETERS', ...)]
)
;
  
```

Simple arguments

EVENT_COLUMN_NAME	Required	This is the name of the event column.
CONVERSION_EVENT_TYPE_VALUE	Required	A list of strings or integers that define the impact events.
EXCLUDING_EVENT_TYPE_VALUE	Required	A list of strings or integers that define the cause events which need to be excluded from the attribution calculation. A row with one of these event type values will be ignored. Note that the excluding event type values can not overlap with the conversion event type values.
OPTIONAL_EVENT_TYPE_VALUE	Required	A list of strings or integers that define the cause events that are optional, which means if there are no other cause events, they will be attributed, otherwise they will be excluded. These values also can not overlap with conversion event and excluding event.
TIMESTAMP_COLUMN_NAME	Required	This is the name of the timestamp column. The column type can be integer, smallint, bigint, timestamp, and time.

Window arguments

WINDOW	Required	<p>This argument specifies the maximum window size used in the attribution calculation. There are three modes:</p> <ul style="list-style-type: none"> • The 'rows:K' mode considers the maximum number of cause events to be attributed, excluding cause events with an event type value specified in the 'excludingEventTypeValue' argument, which means assigning attributions to at most K effective cause events before current impact event. • The 'seconds:K' considers the maximum time difference between current impact event and the earliest effective cause event to be attributed. • And the mixed mode 'rows:K&seconds:K' (two K's are not necessarily the same) considers both constraints and complies to the more strict one.
--------	----------	---

MODEL 1/MODEL 2 arguments

TYPE will be one of:

'SIMPLE'	Using a single distribution model for all the events. In this case, you can specify only one distribution model following the 'TYPE' argument, which is in the format of 'MODEL:PARAMETERS'.
'EVENT_REGULAR'	In this case, you specify a list of 'EVENT:WEIGHT:MODEL:PARAMETERS', which is a list of regular event models. All the weights have to sum to 1.
'EVENT_OPTIONAL'	In this case, you specify a list of 'EVENT:WEIGHT:MODEL:PARAMETERS', which is a list of optional event models. 'EVENT' should be one of those in the 'OPTIONAL_EVENT_TYPE_VALUE' list. All the weights have to sum to 1. Note that when specifying 'OPTIONAL_EVENT_TYPE_VALUE' and 'EVENT_REGULAR', 'EVENT_OPTIONAL' is required.
'SEGMENT_ROWS'	In this case, you specify a list of 'K:WEIGHT:MODEL:PARAMETERS', which is a list of window slices by rows, from the newest (the most recent happened) to the oldest. All the K_i's have to sum to K as specified in the 'rows:K' mode. All the Ks have to sum to the 'K' specified in 'rows:K'. All the weights have to sum to 1.
'SEGMENT_SECONDS'	In this case, you specify a list of 'K:WEIGHT:MODEL:PARAMETERS', which is a list of window slices by seconds, from the newest (the most recent happened) to the oldest. All the K_i's have to sum to K as specified in the 'seconds:K' mode. All the Ks have to sum to the 'K' specified in 'seconds:K'. All the weights have to sum to 1.

The allowed MODEL1/MODEL2 combinations are:

- MODEL1('SIMPLE')
- MODEL1('EVENT_REGULAR')
- MODEL1('EVENT_REGULAR'), MODEL2('EVENT_OPTIONAL') - Note that when specifying 'OPTIONAL_EVENT_TYPE_VALUE' and 'EVENT_REGULAR', 'EVENT_OPTIONAL' is required.
- MODEL1('SEGMENT_ROWS')
- MODEL1('SEGMENT_SECONDS')
- MODEL1('SEGMENT_ROWS'), MODEL2('SEGMENT_SECONDS') - When specify 'rows:K&seconds:K' in the WINDOW argument.

MODEL: This is the model used to calculate the attribution. Currently supported models are 'LAST_CLICK', 'UNIFORM', 'WEIGHTED', 'EXPONENTIAL', 'FIRST_CLICK'. This argument is case sensitive.

PARAMETERS:

- When the distribution model is 'LAST_CLICK', 'UNIFORM', or 'FIRST_CLICK', parameter should be 'NA', otherwise you will get an error.
- When the distribution model is 'WEIGHTED': If it is in 'rows:K' mode, you need to specify a list of weights whose size should be equal to 'K', which is specified in the 'window' argument (or equal to K_i which is specified in corresponding 'K_i:MODEL_i:WEIGHT_i'). Otherwise (in 'seconds:K' mode or in an event model) you can specify as many weights as you want, since they will be dynamically re-normalized. Each weight should be in range [0,1], and all the weights must sum to 1. The weights are specified in the order from left to right as newest to oldest attributed cause event. Semantic of the weights: suppose we have sequence "impression1,impression2,click1,impression3,click2,click3", the window is row based for 3 preceding rows, the user specified weights are "0.5,0.3,0.2". For click1, impression1 has attribution 0.375 (0.3 normalized by 0.3+0.5), impression2 has attribution 0.625 (0.5 normalized by 0.3+0.5). For click2, there is only 1 qualifying cause event: impression3 has attribution 1.0 (0.5 normalized by 0.5).
- When the distribution model is 'EXPONENTIAL', you need to specify a single parameter 'alpha' in range (0,1). Consider the power series: $\omega_i = (1-\alpha) \times \alpha^i$, $0 < \alpha < 1$, $i = 0, 1, \dots$ which sum to 1. These exponential weights can be considered in the same fashion as above (relative weights) when the actual rows being looked back are not infinite.

Examples for Attribution (Single Input)

Example Input Data

The input data for this example is a list of events with user identifiers and timestamps:

Table 3 - 38: Input table (attribution_sample_table)

user_id	event	time_stamp
1	impression	2001-09-27 23:00:01
1	impression	2001-09-27 23:00:03
1	impression	2001-09-27 23:00:05
1	impression	2001-09-27 23:00:07
1	impression	2001-09-27 23:00:09
1	impression	2001-09-27 23:00:11
1	impression	2001-09-27 23:00:13
1	email	2001-09-27 23:00:15
1	impression	2001-09-27 23:00:17
1	impression	2001-09-27 23:00:19
1	click1	2001-09-27 23:00:20
1	optional1	2001-09-27 23:00:21
1	optional2	2001-09-27 23:00:22
1	click2	2001-09-27 23:00:23
2	impression	2001-09-27 23:00:29
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
2	impression	2001-09-27 23:00:43
2	impression	2001-09-27 23:00:47
2	optional	2001-09-27 23:00:49
2	impression	2001-09-27 23:00:51
2	impression	2001-09-27 23:00:53
2	impression	2001-09-27 23:00:55
2	click1	2001-09-27 23:00:59

Some explanation about the input table:

- The 'user_id = 1' partition is mainly used to test the 'rows:K' mode.
- The 'user_id = 2' partition is mainly used to test the 'seconds:K' mode.
- Conversion events are 'click1', 'click2'.
- 'email' serves as potential 'excluding cause event'.
- 'optional', 'optional1', and 'optional2' are potential 'optional cause events'.

Example 1: Event models (with multiple optional event models)

In this example, we specify one distribution model for each type of 'regular' cause event, and one distribution model for each type of optional cause event.

Example 1 SQL-MR Call

```
SELECT * FROM attribution
(
    ON attribution_sample_table
    PARTITION BY user_id
    ORDER BY time_stamp

    EVENT_COLUMN_NAME('event')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2')
    OPTIONAL_EVENT_TYPE_VALUE('optional', 'optional1', 'optional2')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
    MODEL1('EVENT_REGULAR', 'email:0.19:LAST_CLICK:NA',
           'impression:0.81:UNIFORM:NA')
    MODEL2('EVENT_OPTIONAL', 'optional:0.5:UNIFORM:NA',
           'optional1:0.3:UNIFORM:NA', 'optional2:0.2:UNIFORM:NA')
)
ORDER BY user_id, time_stamp;
```

Example 1 Output

Table 3 - 39: Example output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0.09	-19
1	impression	2001-09-27 23:00:03	0.09	-17
1	impression	2001-09-27 23:00:05	0.09	-15
1	impression	2001-09-27 23:00:07	0.09	-13
1	impression	2001-09-27 23:00:09	0.09	-11
1	impression	2001-09-27 23:00:11	0.09	-9
1	impression	2001-09-27 23:00:13	0.09	-7
1	email	2001-09-27 23:00:15	0.19	-5
1	impression	2001-09-27 23:00:17	0.09	-3
1	impression	2001-09-27 23:00:19	0.09	-1
1	click1	2001-09-27 23:00:20		

Table 3 - 39: Example output table (continued)

user_id	event	time_stamp	attribution	time_to_conversion
1	optional1	2001-09-27 23:00:21	0.6	-2
1	optional2	2001-09-27 23:00:22	0.4	-1
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:36	0	
2	impression	2001-09-27 23:00:38	0	
2	impression	2001-09-27 23:00:43	0.2	-16
2	impression	2001-09-27 23:00:47	0.2	-12
2	impression	2001-09-27 23:00:51	0.2	-8
2	impression	2001-09-27 23:00:53	0.2	-6
2	impression	2001-09-27 23:00:55	0.2	-4
2	click1	2001-09-27 23:00:59		

Example 2: Event models (using dynamic weighted distribution model)

In this example, we show that a WEIGHTED distribution model can be used in event models. You can specify as many weights as you want, which determine the maximum number of cause events you want to attribute for an impact event. If the number of effective cause events is larger than the number of weights, then the extra (from oldest) cause events will get zero attribution. If the opposite happens, then the weights will be re-normalized.

Example 2 SQL-MR Call

```
SELECT * FROM attribution
(
    ON attribution_sample_table
    PARTITION BY user_id
    ORDER BY time_stamp

    EVENT_COLUMN_NAME('event')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2')
    OPTIONAL_EVENT_TYPE_VALUE('optional', 'optional1', 'optional2')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
    MODEL1('EVENT_REGULAR', 'email:0.19:LAST_CLICK:NA',
           'impression:0.81:WEIGHTED:0.4,0.3,0.2,0.1')
    MODEL2('EVENT_OPTIONAL', 'ALL:1:WEIGHTED:0.4,0.3,0.2,0.1')
)
ORDER BY user_id, time_stamp;
```

Example 2 Output

Table 3 - 40: Output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0	
1	impression	2001-09-27 23:00:03	0	
1	impression	2001-09-27 23:00:05	0	
1	impression	2001-09-27 23:00:07	0	
1	impression	2001-09-27 23:00:09	0	
1	impression	2001-09-27 23:00:11	0.081	-9
1	impression	2001-09-27 23:00:13	0.162	-7
1	email	2001-09-27 23:00:15	0.19	-5
1	impression	2001-09-27 23:00:17	0.243	-3
1	impression	2001-09-27 23:00:19	0.324	-1
1	click1	2001-09-27 23:00:20		
1	optional1	2001-09-27 23:00:21	0.428571	-2
1	optional2	2001-09-27 23:00:22	0.571429	-1
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:36	0	
2	impression	2001-09-27 23:00:38	0	
2	impression	2001-09-27 23:00:43	0	
2	impression	2001-09-27 23:00:47	0.1	-12
2	impression	2001-09-27 23:00:51	0.2	-8
2	impression	2001-09-27 23:00:53	0.3	-6
2	impression	2001-09-27 23:00:55	0.4	-4
2	click1	2001-09-27 23:00:59		

Comments on Example 2

Please note:

- See click1 in 'user_id = 1'. Based on 'rows:10' mode, the one 'email' and all nine 'impression's are effective. According to the event models 'email:LAST_CLICK:0.19' and 'impression:WEIGHTED:0.81', all 'email's get 0.19 attribution, and all 'impression's get 0.81 attribution. It is clear for 'email' here, while for 'impression's, since we use a WEIGHTED distribution with weights '0.4:0.3:0.2:0.1', so only the newest four 'impression's are attributed, and all the extra older 'impression's get zero attribution. The analysis for click1 in 'user_id = 2' is similar.
- See click2 in 'user_id = 1'. We use a single WEIGHTED distribution model for all types of optional cause event. Since the number of weights is four but the number of effective cause events is only two, so the weights get re-normalized to $0.4/(0.4+0.3) = 0.571429$, and $0.3/(0.4+0.3) = 0.428571$.

Example 3: Window models

In this example we specify both the 'WINDOW_SEGMENTATION_BY_ROWS' and the 'WINDOW_SEGMENTATION_BY_SECONDS' distribution model lists.

Example 3 SQL-MR Call

```
SELECT * FROM attribution
(
    ON attribution_sample_table
    PARTITION BY user_id
    ORDER BY time_stamp

    EVENT_COLUMN_NAME('event')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2')
    EXCLUDING_EVENT_TYPE_VALUE('email')
    OPTIONAL_EVENT_TYPE_VALUE('optional', 'optional1', 'optional2')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
    MODEL1('SEGMENT_ROWS', '3:0.5:EXPONENTIAL:0.5',
           '4:0.3:WEIGHTED:0.4,0.3,0.2,0.1', '3:0.2:FIRST_CLICK:NA')
    MODEL2('SEGMENT_SECONDS', '6:0.5:UNIFORM:NA', '8:0.3:LAST_CLICK:NA',
           '6:0.2:FIRST_CLICK:NA')
)
ORDER BY user_id, time_stamp;
```

Example 3 Output of Attribution

Table 3 - 41: Output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0.2	-19
1	impression	2001-09-27 23:00:03	0	
1	impression	2001-09-27 23:00:05	0.03	-15
1	impression	2001-09-27 23:00:07	0.06	-13

Table 3 - 41: Output table (continued)

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:09	0.09	-11
1	impression	2001-09-27 23:00:11	0.12	-9
1	impression	2001-09-27 23:00:13	0.0714286	-7
1	impression	2001-09-27 23:00:17	0.142857	-3
1	impression	2001-09-27 23:00:19	0.285714	-1
1	click1	2001-09-27 23:00:20		
1	optional1	2001-09-27 23:00:21	0.333333	-2
1	optional2	2001-09-27 23:00:22	0.666667	-1
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:36	0	
2	impression	2001-09-27 23:00:38	0	
2	impression	2001-09-27 23:00:43	0.2	-16
2	impression	2001-09-27 23:00:47	0	
2	impression	2001-09-27 23:00:51	0.3	-8
2	impression	2001-09-27 23:00:53	0.25	-6
2	impression	2001-09-27 23:00:55	0.25	-4
2	click1	2001-09-27 23:00:59		

Comments on Attribution Example 3

- In this example we excluded the 'email'.
- For 'user_id = 1', the 'rows:K' mode was triggered, and the 'WINDOW_SEGMENTATION_BY_ROWS' models were applied to compute attributions.
- For 'user_id = 2', the 'seconds:K' mode was triggered, and the 'WINDOW_SEGMENTATION_BY_SECONDS' models were applied to compute attributions. The three segmentation windows are (from newest to oldest): [58,53], [52,45], [44,39].

Additional Sample Input Data for Single Input Examples

Here we introduce more sample data, in the input table “attribution_sample_table2.” We’ll use this data in examples 4 and 5.

Table 3 - 42: Example input table:

user_id	event	time_stamp
1	impression	2001-09-27 23:00:07
1	impression	2001-09-27 23:00:09
1	impression	2001-09-27 23:00:11
1	impression	2001-09-27 23:00:13
1	email	2001-09-27 23:00:15
1	impression	2001-09-27 23:00:17
1	impression	2001-09-27 23:00:19
1	click1	2001-09-27 23:00:21
1	click2	2001-09-27 23:00:23
2	impression	2001-09-27 23:00:29
2	impression	2001-09-27 23:00:31
2	impression	2001-09-27 23:00:33
2	impression	2001-09-27 23:00:47
2	impression	2001-09-27 23:00:51
2	impression	2001-09-27 23:00:53
2	impression	2001-09-27 23:00:55
2	click1	2001-09-27 23:00:59

Example 4: A single-window model

Example 4 SQL-MR Call

```
SELECT * FROM attribution
(
    ON attribution_sample_table2
    PARTITION BY user_id
    ORDER BY time_stamp

    EVENT_COLUMN_NAME('event')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2')
    EXCLUDING_EVENT_TYPE_VALUE('email')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
    MODEL1('SIMPLE', 'UNIFORM:NA')
)
ORDER BY user_id, time_stamp;
```

Example 4 Output

Table 3 - 43: Example output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:07	0.166667	-14
1	impression	2001-09-27 23:00:09	0.166667	-12
1	impression	2001-09-27 23:00:11	0.166667	-10
1	impression	2001-09-27 23:00:13	0.166667	-8
1	impression	2001-09-27 23:00:17	0.166667	-4
1	impression	2001-09-27 23:00:19	0.166667	-2
1	click1	2001-09-27 23:00:21		
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:47	0.25	-12
2	impression	2001-09-27 23:00:51	0.25	-8
2	impression	2001-09-27 23:00:53	0.25	-6
2	impression	2001-09-27 23:00:55	0.25	-4
2	click1	2001-09-27 23:00:59		

Example 5: Not all segment windows are used

In this example, we show that in case that not all segment window models are used, certain re-normalization scheme will come into play to ensure that all attributions add up to 1 for one impact event.

Example 5 SQL-MR Call

```
SELECT * FROM attribution
(
    ON attribution_sample_table2
    PARTITION BY user_id
    ORDER BY time_stamp
    EVENT_COLUMN_NAME('event')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2')
    TIMESTAMP_COLUMN_NAME('time_stamp')
    WINDOW('rows:10&seconds:20')
    MODEL1('SEGMENT_ROWS', '3:0.5:EXPONENTIAL:0.5',
           '4:0.3:WEIGHTED:0.4,0.3,0.2,0.1', '3:0.2:FIRST_CLICK:NA')
    MODEL2('SEGMENT_SECONDS', '6:0.5:UNIFORM:NA', '8:0.3:LAST_CLICK:NA',
           '6:0.2:FIRST_CLICK:NA')
)
ORDER BY user_id, time_stamp;
```

Example 5 Output

Table 3 - 44: Example output table

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:07	0.0375	-14
1	impression	2001-09-27 23:00:09	0.075	-12
1	impression	2001-09-27 23:00:11	0.1125	-10
1	impression	2001-09-27 23:00:13	0.15	-8
1	email	2001-09-27 23:00:15	0.0892857	-6
1	impression	2001-09-27 23:00:17	0.178571	-4
1	impression	2001-09-27 23:00:19	0.357143	-2
1	click1	2001-09-27 23:00:21		
1	click2	2001-09-27 23:00:23		
2	impression	2001-09-27 23:00:29	0	
2	impression	2001-09-27 23:00:31	0	
2	impression	2001-09-27 23:00:33	0	
2	impression	2001-09-27 23:00:47	0	
2	impression	2001-09-27 23:00:51	0.375	-8
2	impression	2001-09-27 23:00:53	0.3125	-6
2	impression	2001-09-27 23:00:55	0.3125	-4
2	click1	2001-09-27 23:00:59		

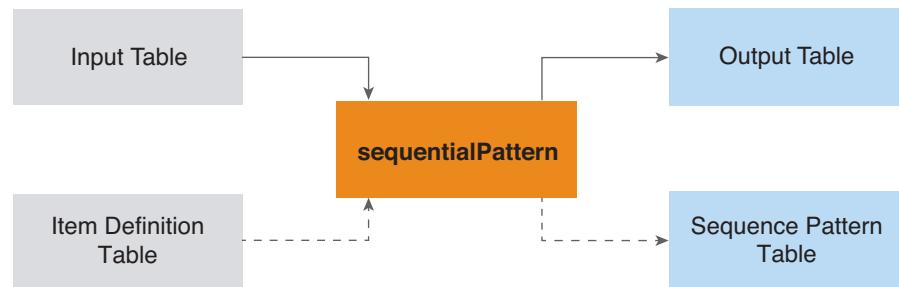
Comments on Attribution Example 5

- For 'user_id = 1', the 'rows:K' mode was triggered, so the 'WINDOW_SEGMENTATION_BY_ROWS' model list was used. But there are only seven rows before click1, so only the first two distribution models, '3:EXPONENTIAL:0.5' and '4:WEIGHTED:0.3' were used. In this situation, we need to re-normalize the window weights to $0.5/(0.5+0.3) = 0.625$, and $0.3/(0.5+0.3) = 0.375$. That is almost exactly what we see here: $(0.357143 + 0.178571 + 0.0892857) = 0.625$, and $(0.15 + 0.1125 + 0.075 + 0.0375) = 0.375$.
- For 'user_id = 2', the 'seconds:K' mode was triggered. In this situation, we have rows in segments [58,53] and [52,45], but we do not have any rows in segment [44,39]. So as above, only the first two distribution models, '6:UNIFORM:0.5' and '8:LAST_CLICK:0.3' were used. We can see that similar re-normalization scheme plays here.

FrequentPaths

Summary

FrequentPaths is a function for mining frequent subsequences as patterns in a sequence database. It has broad applications including the analyses of customer purchase behavior, web access patterns, disease treatments, DNA sequences, and so on.



Background

In a sequential pattern mining application, each sequence consists of an ordered list of itemsets, and each itemset contains at least one item. The items are unordered if there are more than one item in one itemset.

If you use a letter in lower case to represent an item, use "()" to enclose items in an itemset and "<>" to enclose itemsets in a sequence, then the sequence "<(a), (b,c), (d)>" is identical to "<(a),(c,b),(d)>" because items in one itemset are unordered, while sequence "<(a),(b,c),(d)>" is different from "<(a),(d),(b,c)>" because the itemsets are ordered. In web click stream analysis, there is only one item in each itemset, and in purchase behavior analysis there might be multiple items in one itemset as a customer might buy more than one item in one shopping session.

In sequential pattern mining, the conditions for a sequence α to be a subsequence of another sequence β are:

- Each itemset a_i in α must be a subset of another itemset b_j in β .
- The a_i elements in α must have the same order as the b_j elements in β .

A more formal definition is: a sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ is called as a subsequence of another sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$ and β a super sequence of α , if there exists integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$.

The support of a sequence α in a sequence dataset SDB is defined as the number of sequences in SDB that contain α (that is, they are super sequences of α). For example, sequence $\alpha_1 = \langle (a) (bc) \rangle$ is the subsequence of sequence 1 & 2 in the sequence dataset, as shown in [Table 3 - 45](#). The support of α_1 in that dataset is 2. Another sequence $\alpha_2 = \langle (a) (c) \rangle$ is the subsequence of sequences 1 to 4, so the support of α_2 is 4.

Table 3 - 45: A sequence dataset

Id	Sequence
1	<(a) (abc) (ac) (d) (cf)>
2	<(ad) (c) (bc) (ae)>
3	<(ef) (ab) (df) (c) (b)>
4	<(e) (g) (af) (c) (b) (c)>

Given sequence dataset SDB and a threshold T, a sequence α is called as a frequent sequential pattern of SDB, if $\text{support}(\alpha) \geq T$. The problem of sequential pattern mining is to find out all the possible frequent sequential patterns given a sequence dataset SDB and a threshold T.

For two frequent sequential patterns α and β , if α is a subsequence of β and $\text{support}(\alpha) = \text{support}(\beta)$, then α is unclosed. On the other side, for a frequent pattern α if there exists no other pattern that is a super sequence of α and has a same support value, then α is closed. Because a subsequence of a frequent pattern must be frequent also, so all the closed patterns construct a more compact representation of the frequent sequential patterns given a threshold T.



Tip: In the current implementation, when the total frequent pattern number is smaller than 10000, the function checks whether every pattern is closed or not and only closed patterns are outputted. When the total frequent pattern number is larger than 10000, the function does not perform the check because of efficiency, and all patterns (both closed and unclosed) will be outputted.

Input

The FrequentPaths function requires one input table that contains a set of sequences. Each row of the input table represents one item of a sequence. The input to this function can be the output of Teradata Aster nPath.

This function supports two types of input tables:

- [Input table with an item column](#)
- [Input table with an item definition table](#)

Input table with an item column

In the first table type, the input table must contain at least 3 columns:

- One column to define the index of each sequence
- One column to define the order of each row in the sequence
- One column to define the item values

Rows with the same sequence ID belong to the same sequence. Also, rows in a sequence with the same time stamp belong to a same itemset.

As shown in the example in [Table 3 - 46](#), the sequenceid column contains the ID of the sequences, the time1 column determines the itemsets in each sequence, and the item column specify the item values.

Table 3 - 46: input table with an item column

time1	Item	sequenceid
1110000	login	0
1112000	camera	0
1112000	lens	0
1200000	logout	0
1203000	login	1
1300000	shoes	1
1301000	logout	1
1302000	login	2
1340000	...	2

Input table with an item definition table

In the second table type, the input table must contain at least two columns: one column defines the sequence and the other column defines the itemsets, as shown in [Table 3 - 47](#).

Table 3 - 47: Input table without an item column

time1	sequenceid	refurl	productprice
1110000	0	http://www.xxx.com/signin	
1112000	0	http://www.xxx.com/camera	1230
1112000	0	http://www.xxx.com/order	
1200000	0	http://www.xxx.com/signout	
1203000	1	http://www.xxx.com/signin	
1300000	1	http://www.xxx.com/book	15
1301000	1	http://www.xxx.com/signout	
1302000	2	http://www.xxx.com/signin	
1340000	2	http://www.xxx.com/signout	

In [Table 3 - 47](#), the item values of each row are determined by a second table ([Table 3 - 48](#)), which is referred to as an item definition table. Each row of the item definition table contains a predicate and an item value. The predicates is applied to each row of the input table. For a row, if one predicate is true, then the value of that row is marked as the one corresponding to the predicate.

There should be at least 3 columns in the item definition table:

- One column determines the index of each predicate.
The data type of the index column should be short, int or long.
- One column gives out the definition of each predicate.
The data type of the definition column should be type of char, varchar, or text.
- One column gives out the value if one predicate is true.
The data type of the value column should be type of char, varchar, or text.

Table 3 - 48: Item definition table example

ID	Definition	Item
1	refurl like '%signin%	LOGIN
2	refurl like '%signout%	LOGOUT
3	productprice > 1000 and refurl like '%camera%	HIGHENDCAM

Note that the function applies the predicates to the input table based on the order of their index. If there is one row for which more than one predicate is true, the function assigns the row the value corresponding to the predicate with the smallest index. The function does not process the rows in the input table that have no corresponding definition in the definition table.

Output

[Table 3 - 49](#) defines the schema of the output table generated by this function. The id column defines the index (starting from 0) of the output patterns, the pattern column specifies the patterns in string format, and the support column specifies the support value of each pattern in the input table.

Table 3 - 49: Output schema

column	type
id	int
pattern	varchar
support	int

The output patterns are in this format:

item1,item2;item3,item4;....;itemn;

And if the “SEQUENCEPATTERNRELATION” clause is specified, a table with a given name in that clause is created, which stores a list of (sequence_id,pattern_id) pairs that are the index of the patterns as well as the sequences which contain each pattern.

Usage

Syntax (version 1.1)

```
SELECT *
FROM FrequentPaths(
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    PARTITIONCOLUMNS('partition_column_list')
    [TIMECOLUMN('sort_column_name')]
    [PATHFILTERS('filter_list')]
    [GROUPBYCOLUMNS(groupby_column_list)]
    [SEQUENCEPATTERNRELATION('sequence_pattern_table_name')]
    [ITEMCOLUMN('sequence_column_name')]
    [ITEMDEFINITION('item_definition_table:
        [id_column:definition_column:item_column]')]
    [PATHCOLUMN('path_column_name')]
    MINSUPPORT('minimum_support_value')
    [MAXLENGTH('maximum_length')]
    [MINLENGTH('minimum_length')])
);
```

Arguments

<i>DOMAIN</i>	Optional	The IP address of the queen node. Default domain is queen of the current cluster.
<i>DATABASE</i>	Optional	The Name of the database where the input table is present. Default database is 'beehive'.
<i>USERID</i>	Optional	The Aster Database database user name of the user. Default userid is 'beehive'.
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	The sequence table, where each row of record is one item belonging to a sequence. If the <i>INPUTTABLE</i> clause does not contain schema information, the function assumes that it is located in the public schema. Rows with NULL-value cells are ignored.
<i>PARTITIONCOLUMNS</i>	Required	A list of columns that provide the partition keys of a sequence.
<i>TIMECOLUMN</i>	Optional	(Required when <i>ITEMCOLUMN</i> or <i>ITEMDEFINITION</i> is specified) A column in <i>INPUTTABLE</i> that determines the order of items in a sequence. Items with a same time value in a sequence belong to one itemset. This column must not be of type real or double.

PATHFILTERS	<i>Optional</i>	<p>A list of filters each of which comprises a set of constraints. You can define three constraints:</p> <ul style="list-style-type: none"> • STW(A, B, ...)—Start-with constraint, where A, B are items and are separated by a comma. This constraint requires that the first itemset of the input sequence must contain at least one of the specified items. For example, STW(c,d) requires that the input sequence must start with item “c” or “d”. Sequence “(a, c), e, (f, d)” meets this constraint because the first itemset “(a,c)” contains item “c”, while “e, (f,d)” does not meet this constraint. • EDW(A, B, ...)—End-with constraint, where A, B are items and are separated by a comma. This constraint requires that the last itemset of the input sequence must contain at least one of the specified items. For example, EDW(f,g) requires that the input sequence must end with item “f” or “g”. Sequence “(a, b), e, (f, d)” meets this constraint because the last itemset “(f,d)” contains item “f”, while “(a,b),e” does not meet this constraint. • CTN(A, B, ...)—Containing constraint, where A, B are items and are separated by a comma. This constraint requires that a sequence must contain at least one of the specified items. For example, CTN(a,b) requires that the input sequence must contains “a” or “b”. So, the sequence “(a,c), d, (e,f)” meets this constraint while “d, (e,f)” does not meet it. <p>A filter can contain multiple constraints, which should be separated by space. If a filter comprises more than one constrain, all the constraints take effect, which means that the input sequence should meet all of the constraints. An example of such a filter is 'STW(c,d) EDW(g,k) CTN(e)', which requires the input sequence to start with item “c” or “d”, end with “g” or “k”, and contain “e”. Note that a constraint type must appear at most once in one filter because all constraints defined in a filter take effect simultaneously. For example, filter 'STW(c,d) STW(e,h)' is an illegal input.</p> <p>The PATHFILTERS argument can have multiple filters. Sequences in the input table that meet the constraints of at least one of the filters are passed to the function as input. Sequences that do not meet any of the filters are filtered out.</p> <p>In a default case, the items in one constraint are separated by commas. However, you can define your own separator symbols through a SEPARATOR(<i>symbol</i>) clause. The SEPARATOR clause specifies the separator symbol used for all constraints in the PATHFILTER argument, and must be defined in an isolated string in the arguments list. An example is:</p> <pre>PATHFILTERS('SEPARATOR(#)', 'STW(c#d) EDW(g#k) CTN(e)', 'CTN(h#k)')</pre>
GROUPBYCOLUMNS	<i>Optional</i>	<p>A list of columns in the format of ('<i>col1</i>', '<i>col2</i>', ..., '<i>coln</i>'). The input table should be first grouped by the groupby columns, and then the sequential pattern mining for each group is performed separately. The patterns and support values are computed just inside the group. In this case, the groupby columns will be accumulated to the output table. The groupby columns must not be of type real nor double.</p>

<i>SEQUENCEPATTERNR</i>	Optional <i>ELATION</i>	<p>The name of table in which to store the sequence-pattern pairs. If this clause is specified, a table with the specified name is created to store a list of sequence:pattern pairs, which means that the sequence in the input table contains the pattern in the output.</p> <p>The schema of this table is:</p> <ul style="list-style-type: none"> • <i>partition_column_name</i> • <i>pattern_id</i> <p>The 1st column inherits the value of the <i>partitionColumn</i> clause along with the data type of the column.</p> <p>The 2nd column is same as the 1st column in the output schema.</p> <p>For example, assume that one sequence has a partition value of "1" and it contains 3 patterns with IDs 2, 9, and 10. In that case, there will be three tuples in the table: ("1", 2), ("1", 9), and ("1", 10).</p> <p>If the <i>SEQUENCEPATTERNRELATION</i> clause doesn't contain schema information, the table is created in the "public" schema.</p> <p>If the output pattern number is zero, then the <i>sequencepatternrelation</i> table is not created.</p>
<i>ITEMCOLUMN</i>	Optional	The column names of the items in <i>INPUTTABLE</i> .
<i>ITEMDEFINITION</i>	Optional	<p>The table name, index column name, definition column name, and item column name of the item definition table. If this clause is used, the item value of each row in the input table is determined according to the predicates defined in the item definition table (see Table 3 - 48).</p> <p>The item definition table should have at least three columns: an index column (should be of type integer), a definition column that defines the predicates (should be of type char, varchar, or text), and an item column that defines the item names if the corresponding predicate is true (should be of type char, varchar, or text)</p> <p>If the <i>ITEMDEFINITION</i> clause doesn't contain schema information, the function assumes that it is located in the "public" schema.</p> <p>You must choose either the <i>ITEMCOLUMN</i> or <i>ITEMDEFINITION</i> clause to determine the item for each row.</p>
<i>PATHCOLUMN</i>	Optional	A column where each row is a sequence string in the format of '[A, B, C, D]' where A, B, C, D are items separated by commas. This sequence string can be outputted by the accumulate function in Teradata Aster nPath. If you specify this clause, it is required for one itemset to have only one item. Use this clause exclusively with the <i>ITEMCOLUMN</i> and <i>ITEMDEFINITION</i> clauses.

<i>MINSUPPORT</i>	Required	<p>A positive real number that determines the threshold T for deciding whether a sequential pattern is frequent or not. There are two ways to set this argument:</p> <ul style="list-style-type: none"> Assuming the total number of sequences is N, if <i>MINSUPPORT</i> is set to a value ms in the range of $(0,1]$ (that is, $0 < ms \leq 1$), then the threshold for frequent sequential patterns is $T=N*ms$. In this case, ms is treated as a relative threshold. For example, assume there is a total of 1000 sequences in the input table (or view) and you use <i>MINSUPPORT</i>('0.05'), the threshold for frequent sequential patterns is 50. If <i>MINSUPPORT</i> is set to a value ms in the range of $(1,+\infty)$ (that is, $ms > 1$), then the threshold for frequent sequential patterns is $T=ms$. In this case, ms is treated as an absolute threshold. For example, if you use <i>MINSUPPORT</i>('50'), then the threshold for frequent sequential patterns is 50 no matter how many sequences are in the input table (or view). <p>A pattern is considered frequent if it has a support value $\geq T$. Because only frequent patterns are outputted, the <i>MINSUPPORT</i> argument controls the number of output patterns. Normally the processing time increases exponentially if a small threshold is set, therefore, we recommend that you start the trail with a bigger value (for example, 5% of the total sequence number). If you cannot determine the total number of sequences in the table, use a relative threshold, that is set <i>MINSUPPORT</i> to a value smaller than 1 (for example, 0.05).</p> <p>If both the <i>GROUPBYCOLUMNS</i> and <i>MINSUPPORT</i> arguments are specified, and <i>MINSUPPORT</i> specifies a relative value (in the range of $(0,1.0]$), the sequence number N is calculated for each group separately, which means that the thresholds are different for different groups. For example, assuming there are $N1$ sequences in group1 and $N2$ sequences in group2, and the <i>MINSUPPORT</i> value is ms ($0 < ms \leq 1$), then the threshold for group1 is $N1*ms$ and the threshold for group2 is $N2*ms$.</p> <p>If both <i>PATHFILTERS</i> and <i>MINSUPPORT</i> arguments are specified, and <i>MINSUPPORT</i> is a relative value (in the range of $(0,1.0]$), the sequence number N is the number of sequences that meet the constraints of the filters.</p>
<i>MAXLENGTH</i>	Optional	The maximum length of the output sequential patterns. In default cases, there is no upper limitation on the length of patterns. The length of a pattern sequence is defined as the number of itemsets in it. For example, the length of sequence "1,2;3,4,5" is 2.
<i>MINLENGTH</i>	Optional	The minimum length of the outputted sequential patterns. In default cases, the minimum length is 1.

Examples

Example 1: Sample SQL-MR call with the *itemcolumn* clause specified

Input

Table 3 - 50: Input table (testseq)

time1	item	sequenceid
11	A	1
12	A	1

Table 3 - 50: Input table (testseq) (continued)

time1	item	sequenceid
12	B	1
12	C	1
13	A	1
13	C	1
14	D	1
15	C	1
15	F	1
21	A	2
21	D	2
22	C	2
23	B	2
23	C	2
24	A	2
24	E	2
31	E	3
31	F	3
32	A	3
32	B	3
33	D	3
33	F	3
34	C	3
35	B	3
41	E	4
42	G	4
43	A	4
43	F	4
44	C	4
45	B	4
46	C	4

Sample SQL-MR Call

This is an example that uses the itemcolumn clause:

```
SELECT * FROM FrequentPaths(
    ON (SELECT 1)
    PARTITION BY 1
    PASSWORD('beehive')
    INPUTTABLE('testseq')
    PARTITIONCOLUMN('sequenceid')
    TIMECOLUMN('time1')
    ITEMCOLUMN('item')
    MINSUPPORT(2)
) ;
```

Output

Table 3 - 51: Output table

id	pattern	support
0	A,B;F	2
1	A;B;C	2
2	D;C;B	2
3	E;B;C	2
4	F;B;C	2
5	A,B;D;C	2
6	A;B,C;A	2
7	E;A;C;B	2
8	E;F;C;B	2
9	E	3
10	F	3
11	B;C	3
12	D;C	3
13	A;C;B	3
14	A;C;C	3
15	A;B	4
16	A;C	4

Example 2: Sample SQL-MR call with the itemdefinition clause

specified***Input***

Input table click_log:

Table 3 - 52: Input table (click_log)

url	access_time	user_id
https://www.google.com/	201211061013	0
http://www.ebay.com/	201211061013	0
www.youtube.com	201211061014	0
http://www.yahoo.com	201211061013	0
https://www.google.com/	201211062000	1
www.youtube.com	201211062100	1
http://www.amazon.com	201211062100	1
https://www.google.com/	201211062201	2
http://www.yahoo.com	201211062202	2
http://www.amazon.com	201211062202	2
http://www.baidu.com	201211062203	2

Item definition table:

Table 3 - 53: Item definition table (url_def)

id	definition	item
1	url like '%google%'	google
2	url like '%yahoo%'	yahoo
3	url like '%ebay%'	ebay
4	url like '%youtube%'	youtube
5	url like '%amazon%'	amazon
6	url like '%baidu%'	baidu

Sample SQL-MR Call

Example that uses the itemdefinition clause:

```
SELECT * FROM FrequentPaths(
    ON (SELECT 1) PARTITION BY 1 PASSWORD('beehive')
    INPUTTABLE('click_log')
    PARTITIONCOLUMN('user_id')
    TIMECOLUMN('access_time')
    ITEMDEFINITION('url_def:[id:definition:item]')
    MINSUPPORT(2)) ;
```

Output

Table 3 - 54: Output table

id	definition	support
0	yahoo	2
1	google;amazon	2
2	google;youtube	2
3	google	3

Example 3: SQL-MR call with groupbycolumns clause specified***Input***

Table 3 - 55: Input table (testseqg)

time1	Item	sequenceid	grp
11	A	1	1
12	A	1	1
12	B	1	1
12	C	1	1
13	A	1	1
13	C	1	1
14	D	1	1
15	C	1	1
15	F	1	1
21	A	2	1
21	D	2	1
22	C	2	1
23	B	2	1
23	C	2	1
24	A	2	1
24	E	2	1
31	E	3	2
31	F	3	2
32	A	3	2
32	B	3	2

Table 3 - 55: Input table (testseqg) (continued)

time1	Item	sequenceid	grp
33	D	3	2
33	F	3	2
34	C	3	2
35	B	3	2
41	E	4	2
42	G	4	2
43	A	4	2
43	F	4	2
44	C	4	2
45	B	4	2
46	C	4	2

SQL-MapReduce Call

Example that uses the groupbycolumns clause:

```
SELECT * FROM FrequentPaths (
    ON (SELECT 1)
    PARTITION BY 1
    PASSWORD('beehive')
    INPUTTABLE('testseqg')
    PARTITIONCOLUMN('sequenceid')
    GROUPBYCOLUMNS('grp')
    TIMECOLUMN('time1')
    ITEMCOLUMN('item')
    MINSUPPORT(2)
);
```

Output

Table 3 - 56: Output table

id	pattern	support	grp
0	D;C	2	1
1	A;C;C	2	1
2	A;B,C;A	2	1
3	E;B;C	2	2
4	F;B;C	2	2
5	E;A;C;B	2	2
6	E;F;C;B	2	2

Example 4: SQL-MR call with the sequencepatternrelation clause specified

Table 3 - 57: input table (testseq)

time1	Item	sequenceid
11	A	1
12	A	1
12	B	1
12	C	1
13	A	1
13	C	1
14	D	1
15	C	1
15	F	1
21	A	2
21	D	2
22	C	2
23	B	2
23	C	2
24	A	2
24	E	2
31	E	3
31	F	3
32	A	3
32	B	3
33	D	3
33	F	3
34	C	3
35	B	3
41	E	4
42	G	4
43	A	4
...

SQL-MapReduce Call

Example with the sequencepatternrelation clause used

```
SELECT * FROM FrequentPaths (
    ON (SELECT 1)
    PARTITION BY 1
    PASSWORD('beehive')
    INPUTTABLE('testseq')
    PARTITIONCOLUMN('sequenceid')
    TIMECOLUMN('time1')
    ITEMCOLUMN('item')
    MINSUPPORT(2)
    SEQUENCEPATTERNRELATION('sp_table')
) ;
```

Output

The output table:

Table 3 - 58: Output table

id	pattern	support
0	A,B;F	2
1	A;B;C	2
2	D;C;B	2
3	E;B;C	2
4	F;B;C	2
5	A,B;D;C	2
6	A;B,C;A	2
7	E;A;C;B	2
8	E;F;C;B	2
9	E	3
10	F	3
11	B;C	3
12	D;C	3
13	A;C;B	3
14	A;C;C	3
15	A;B	4
16	A;C	4

The sequence pattern table:

Table 3 - 59: sp_table

sequenceid	pattern_id
3	0
1	0
1	1
4	1
3	2
2	2
3	3
4	3
3	4
4	4
1	5
3	5
1	6
2	6
...	...

Example 5: SQL-MR Call with the Pathfilters Clause Specified

Input Table

Table 3 - 60: testseqq

tm	item	seqid1	seqid2	grp
11	A	1	1	1
12	A	1	1	1
12	B	1	1	1
12	C	1	1	1
13	A	1	1	1
13	C	1	1	1
14	D	1	1	1
15	C	1	1	1
15	F	1	1	1
41	E	1	2	2
42	G	1	2	2
43	A	1	2	2
43	F	1	2	2
44	C	1	2	2
45	B	1	2	2
46	C	1	2	2
21	A	2	1	1
21	D	2	1	1
22	C	2	1	1
23	B	2	1	1
23	C	2	1	1
24	A	2	1	1
24	E	2	1	1
31	E	2	2	2
31	F	2	2	2
32	A	2	2	2
32	B	2	2	2
33	D	2	2	2

Table 3 - 60: testseqg (continued)

tm	item	seqid1	seqid2	grp
33	F	2	2	2
34	C	2	2	2
35	B	2	2	2

Example SQL-MR call:

```
SELECT *
FROM FrequentPaths(
    ON (SELECT 1)
    PARTITION BY 1
    PASSWORD('beehive')
    INPUTTABLE('testseqg')
    PARTITIONCOLUMNS('seqid1','seqid2')
    TIMECOLUMN('tm')
    ITEMCOLUMN('item')
    PATHFILTERS('STW(A) EDW(E,F)')
    MINSUPPORT(2)
    SEQUENCEPATTERNRELATION('sp_table')
) order by id;
```

Output

Table 3 - 61: Example output

id	pattern	support
0	D;C	2
1	A;C;C	2
2	A;B,C;A	2

Using Teradata Aster nPath and FrequentPaths to Select Sequences

In the Teradata Aster nPath function, you can define a pattern to find matching sequences in the input data. Thus, you could use the Teradata Aster nPath function to select sequences of interest before running the FrequentPaths function.

For example, consider this input table:

Table 3 - 62: sequence_table

id	tm	item
1	11	A
1	12	B
1	13	C
2	21	B
2	22	C

Table 3 - 62: sequence_table

id	tm	item
2	23	D
3	31	A
3	32	D
3	33	C

To find sequences that start from “A” and end with “C”, run this SQL-MR Teradata Aster nPath call in which the accumulate function is used to output the full sequence.

```
create view nPath_output as (
    select * from npath(
        on sequence_table partition by id order by tm
        pattern('itemA.itemAny*.itemC')
        symbols(item='A' as itemA, item='C' as itemC, true as itemAny)
        result(first(id of itemA) as id,
               accumulate(item of any(itemA, itemAny, itemC)) as path)
        mode(NONOVERLAPPING))
);
```

Then, run this query to display the sequences found in the input data:

```
select * from nPath_output;
```

Table 3 - 63: Teradata Aster nPath output

id	item
1	[A, B, C]
3	[A, D, C]

Note that in [Table 3 - 63](#), the sequence strings in the path column are in the format required by the PATHCOLUMN argument of the FrequentPaths function. The next step is to run the FrequentPaths function to find the patterns in the subset of sequences that start from A and end with C:

```
select * from FrequentPaths (
    on (select 1) partition by 1
    PASSWORD('beehive')
    INPUTTABLE('nPath_output')
    PARTITIONCOLUMNS('id')
    PATHCOLUMN('path')
    MINSUPPORT('2')
);
```

[Table 3 - 64](#) displays the results.

Table 3 - 64: Output of FrequentPaths

id	pattern	support
0	A;C	2

Error Messages

You might see these errors:

- ERROR 1: Illegal schema or table name. Found: table_ "name
REASON: The input relation name is not in correct format;
- ERROR 2: 'TIMECOLUMN' clause could not be assigned a same column as 'PARTITIONCOLUMNS'
REASON: The 'TIMECOLUMN' clause is specified a column which is same as 'PARTITIONCOLUMNS' clause;
- ERROR 3: 'TIMECOLUMN' clause could not be assigned a same column as 'GROUPBYCOLUMNS'
REASON: The 'TIMECOLUMN' clause is specified a column which is already in 'GROUPBYCOLUMNS' clause;
- ERROR 4: 'PARTITIONCOLUMNS' clause could not be assigned a same column as 'GROUPBYCOLUMNS'
REASON: The 'PARTITIONCOLUMNS' clause is specified a column which is already in 'GROUPBYCOLUMNS' clause;
- ERROR 6: 'ITEMCOLUMN' clause could not be assigned a same column as 'PARTITIONCOLUMNS'
REASON: The 'ITEMCOLUMN' clause is specified a column which is same as 'PARTITIONCOLUMNS' clause;
- ERROR 7: 'ITEMCOLUMN' clause could not be assigned a same column as 'TIMECOLUMN'
REASON: The 'ITEMCOLUMN' clause is specified a column which is same as 'TIMECOLUMN' clause;
- ERROR 8: 'ITEMCOLUMN' clause could not be assigned a same column as 'GROUPBYCOLUMNS'
REASON: The 'ITEMCOLUMN' clause is specified a column which is already in 'GROUPBYCOLUMNS' clause;
- ERROR 9: Either 'ITEMCOLUMN' or 'ITEMDEFINITION' clause should be chosen to determine the items.
REASON: Neither 'ITEMCOLUMN' nor 'ITEMDEFINITION' is specified so the function can not determine the item values
- ERROR 10: 'ITEMCOLUMN' and 'ITEMDEFINITION' clause cannot be chosen simultaneously
REASON: Both 'ITEMCOLUMN' and 'ITEMDEFINITION' are specified simultaneously.
- ERROR 11: 'MINSUPPORT' should be a float value in the range (0.0,1.0] or a integer value larger than 1. Found: XXX
REASON: 'MINSUPPORT' should be a float value in the range (0.0,1.0] or a integer value larger than 1
- ERROR 12: 'MINSUPPORT' could not be an infinite value
REASON: 'Infinity' is set to 'MINSUPPORT' clause

- ERROR 13: 'MINLENGTH' must be a positive integer. Found: XXX
REASON: 'MINLENGTH' is specified a negative integer or 0, or a non-integer value is set to this clause.
- ERROR 14: 'MAXLENGTH' must be a positive integer. Found: XXX
REASON: 'MAXLENGTH' is specified a negative integer or 0, or a non-integer value is set to this clause.
- ERROR 15: 'MAXLENGTH' value should be larger than 'MINLENGTH'. Found: maxlen_length_value, minlength_value
REASON: The value set to 'MINLENGTH' is larger than the value set to 'MAXLENGTH' clause.
- ERROR 16: Relation "table_name" does not exist
REASON: The required "table_name" relation does not exist.
- ERROR 17: Column "column_name" does not exist in relation "relation_name"
REASON: The specified column does not exist in the given relation.
- ERROR 18: The column "column_name" in relation "relation_name" should not be of type real nor double
REASON: For 'TIMECOLUMN', 'GROUPBYCOLUMNS' clauses, we require the specified columns should not be type of real nor double.
- ERROR 19: Relation "sequencepatternrelation_table_name" already exists
REASON: This error message is thrown if a the relation specified in the 'SEQUENCEPATTERNRELATION' clause already exists.
- ERROR 20: 'GROUPBYCOLUMNS' clause contains repeated column names: "groupby_column_list"
REASON: Duplicate column names in 'GROUPBYCOLUMNS' clause
- ERROR 21: Error definition at id XXX in table "itemdefinition_table_name": "definition_string"
REASON: The definition in item definition table is not a correct SQL predicate
- ERROR 22: No valid definitions exist in relation "itemdefinition_table_name"
REASON: No definition exists in the item definition table.
- ERROR 23: The column "column_name" in relation "table_name" has to be of type char, varchar or text.
REASON: The definition and item column in the item definition table is required to be of type char, varchar or text.
- ERROR 24: The column "column_name" in relation "table_name" has to be of type integer.
REASON: The id column in the item definition table must be of type integer.
- ERROR 25: The "ITEMDEFINITION" should be in the format of "ITEMDEFINITION('<items_definition_table>:[id_column:item_definition_column:item_column]')". Found: "input_string"
REASON: The parameter set to 'ITEMDEFINITION' clause is not in correct format.

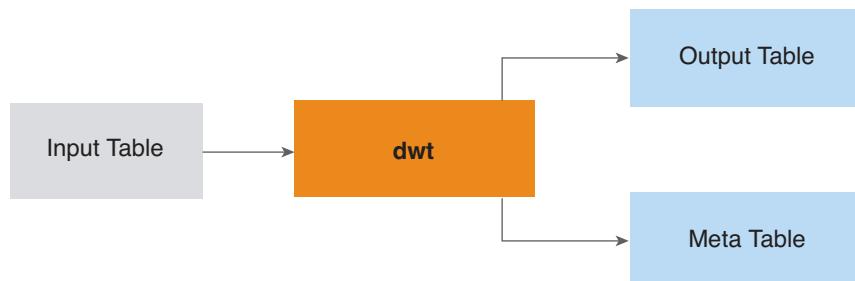
- ERROR 26: The column "column_name" in table "table_name" should be one of the following types: int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, bytea
REASON: The column specified in 'PARTITIONCOLUMN' clause should be a legal partition key type.
- ERROR 27: The items are too big to be stored in memory ()
REASON: The function will store all item values in memory. This error will be reported if the size of memory for storing the item values exceed 90% of maximum memory of Java JVM.

DWT

Summary

This function implements Mallat's algorithm which is an iterate algorithm in the Discrete Wavelet Transform (DWT) field, and is designed to apply wavelet transform on multiple sequences simultaneously. The input is typically a set of time series sequences. Each sequence consists of values in a specific order. You specify the wavelet name, transform level, and optionally the extension mode. The result is the transformed sequences in Hilbert space with corresponding component identifier and index. The transformation is also called decomposition.

The result can be filtered to reduce the length of the sequence. And the original sequence can be reconstructed according to the result via IDWT. Thus the function can be adopted in compression and denoise.



Background

DWT is a kind of time-frequency analysis tool for which the wavelets are discretely sampled. DWT is different from the Fourier transform, which provides frequency information on the whole time domain. A key advantage of DWT is that it provides frequency information at different time points.

Mallat's algorithm can be described as several iterative steps. For example, in the case of a 3-level wavelet transform, the algorithm performs these steps:

- 1 Use $S(n)$ as the original time domain sequence as the input of level 1.
- 2 The input sequence is convolved with high-pass filter $h(n)$ and low-pass filter $g(n)$, then the convolved sequences are downsampled respectively.
Two sequences are generated. The generated sequences are the detail coefficients Dk and the approximation coefficients Ak in level k .
- 3 If current level k reaches max transform level n , then go to the end. Otherwise, Ak is used as the input sequence for the next level; increase the current level k by 1, then go to step 2.

Usage

Syntax (version 1.0)

DWT is adopted to emit a list of coefficients and some meta data for each sequence. The function assumes that each sequence can be fitted into the memory of the worker.

```
SELECT * FROM dwt(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    [PASSWORD('password')]
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    METATABL('meta_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
    [PARTITIONCOLUMNS('partition_column_name1',
        'partition_column_name2', ..., 'partition_column_nameN')]
    WAVELETNAME('wavelet_name') |
    WAVELETFILTERTABLE('wavelet_filter_table_name')
    LEVEL(level)
    [EXTENSIONMODE('<extension mode>')]
) ;
```

Arguments

DOMAIN	Optional	Has the form, <i>host:port</i> . The host is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: <code>[::1]:2406</code> . The port is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
DATABASE	Optional	This is the name of the database where the input table is present. Default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
PASSWORD	Required	The Aster Database password of the user.
INPUTTABLE	Required	The name for the input relation. It should be a table or a view. Each row of the relation consists of sample data and corresponding sequence id and time metric. The inputtable should contain no more than 1594 columns. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.

<i>OUTPUTTABLE</i>	Required	The name for the output table for the coefficients generated by the wavelet transform. This table must not exist, so if it does exist, you must DROP it before running the function again. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
<i>METATABLE</i>	Required	The name for the output table for the meta information of the transformation. This table must not exist, so if it does exist, you must drop it before running the function again. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
<i>INPUTCOLUMNS</i>	Required	<p>The data columns of the input table to be transformed. You can either explicitly list all the names, for example, INPUTCOLUMNS('col1','col2', ...), or specify a range of columns, for example, INPUTCOLUMNS('[4:33]'), or some combination of the above, for example, INPUTCOLUMNS('col1',[4:21],[25:53],'col73').</p> <p>Ranges are specified using this syntax: "[<start_column>:<end_column>]", and the column index starts from 0. These columns must contain numeric values between -1e308 and 1e308. If NULL exists in the columns, it is treated as zero.</p>
<i>SORTCOLUMN</i>	Required	The column that represents the order of samples in a specific sequence. In a time series sequence, the column can consist of timestamp values. If <i>SORTCOLUMN</i> has duplicate elements in a specific sequence (partition), it might cause different transform result for the sequence because the order of the sequence is not guaranteed.
<i>PARTITIONCOLUMNS</i>	Optional	One or more columns identify the different sequences. Every row that has the same value in <i>PARTITIONCOLUMNS</i> belongs to the same sequence. If not specified, all the rows are treated as one sequence and then a column named "dwt_id**"(* represents a random name), filled with value 1, is auto generated in the outputtable and metatable as the distribute key. If multiple columns are provided, the first one is adopted as the distribute key of the output tables and thus, it should have one of these types: int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea.
<i>WAVELETFILTERTABLE</i>	Optional	The table that stores the coefficients of the filters. The column definitions of the table are listed Table 3 - 65 . The data read from the table is used in the wavelet transform. If schema is not specified in the argument, "public" is added to the argument as schema name automatically.
<i>WAVELETNAME</i>	Optional	The name of predefined wavelet filter to be used. The supported predefined wavelet names are listed in Table 3 - 66 .
<i>LEVEL</i>	Required	The level of wavelet transform to be performed. Must be a positive integer in the range of [1,1000].
<i>EXTENSIONMODE</i>	Optional	The method for handling the problem of border distortion. Currently supported values are: "sym" (replicates boundary value symmetrically), "zpd"(zero-padding), and "ppd"(periodic extension). The default value is "sym".For more information, see Table 3 - 67 .

Limitations

You must choose either *WAVELETFILTERTABLE* or *WAVELETNAME* to specify the wavelet used in transformation. If both of the parameters are specified, an exception is thrown.

All the schema names, table names, and column names are treated as case-insensitive arguments in the function. If either the schema name, the table name, the column name, or

both schema and table names include capital characters, you must individually surround each name with double-quotation marks.

WAVELETFILTERTABLE definition

Table 3 - 65: WAVELETFILTERTABLE definition

filtername	filtervalue
lowpassfilter	The low-pass filter in decomposition, represented in comma separated sequence, it is the conjugated scale coefficients for orthogonal wavelet. For example, -0.1294095225512604,0.2241438680420134,0.8365163037378081,0.4829629131445342
highpassfilter	The high-pass filter in decomposition, represented in comma separated sequence, it is the conjugated wavelet coefficients for orthogonal wavelet. For example, -0.4829629131445342,0.8365163037378081,-0.2241438680420134,-0.1294095225512604
ilowpassfilter	The low-pass filter in reconstruction, represented in comma separated sequence, it is the scale coefficients for orthogonal wavelet. For example, 0.4829629131445342,0.8365163037378081,0.2241438680420134,-0.1294095225512604
ihighpassfilter	The high-pass filter in reconstruction, represented in comma separated sequence, it is the wavelet coefficients for orthogonal wavelet. For example, -0.1294095225512604,-0.2241438680420134,0.8365163037378081,-0.4829629131445342

Supported wavelet names

Table 3 - 66: Supported wavelet names

wavelet families	wavelet names
Daubechies	'db1' or 'haar', 'db2', ... , 'db10'
Coiflets	'coif1', ... , 'coif5'
Symlets	'sym1', ... , 'sym10'
Discrete Meyer	'dmey'
Biorthogonal	'bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8', 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5'
Reverse Biorthogonal	'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9', 'rbio4.4', 'rbio5.5'

Supported extension modes

Table 3 - 67: Extension modes

extension mode	description	example
sym	default mode, replicate boundary value symmetrically, mirror the points near boundary	4 4 3 2 1 1 2 3 4(original sequence) 4 3 2 1 1
zpd	zero-padding, fill all the boundary value with zero	0 0 0 0 0 1 2 3 4(original sequence) 0 0 0 0 0
ppd	periodic extension, fill boundary values as the input sequence is a periodic one	4 1 2 3 4 1 2 3 4(original sequence) 1 2 3 4 1

The above table assumes that the sequence before the extension is 1 2 3 4 and the kernel length is 6, which means that the length of the sequence should be extended by 5 positions at both directions, respectively.

Output

Returns a message indicating whether the function was successful. The transformed sequences are in the output table and the wavelet-related information is in the meta table.

The schema of the output table consists of the following columns:

Table 3 - 68: Output table schema

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	inherited from input table	The values are inherited from input table for each sequence. If the argument PARTITIONCOLUMNS is not specified, a column named "dwt_id**" (* represents random name), filled with the integer value 1, is used as the partition column.
waveletid	int	The index of each wavelet coefficient (starting from 1 for each sequence).
waveletcomponent	varchar	The component that the coefficient belongs to. Possible values are An Dn Dn-1... D1, where n is the level of the transformation.
INPUTCOLUMNS (names are inherited from input table)	double	The coefficients after the wavelet transform of the corresponding input column.

The schema of the meta table consists of following columns:

Table 3 - 69: Meta table schema

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	inherited from input table	The values are inherited from input table for each sequence. If argument PARTITIONCOLUMNS is not specified, a column named 'dwt_id**' (the * character represents a random name), filled with the integer value 1, is used as the partition column.
meta	varchar	The name of the specified meta information. The possible meta names are listed in Table 3 - 70 .
content	varchar	The value of the corresponding meta name.

For each sequence, this information is listed in the meta table:

Table 3 - 70: Sequence information

meta	content
blocklength	The length of each component after transformation, from An to D1. For example, 8,8,13.
length	The length of the sequence before the transformation. For example, 24.
waveletname	The name of the wavelet used in the transformation. For example, db2.
lowpassfilter	The low-pass filter in the decomposition of the wavelet used in DWT.
highpassfilter	The high-pass filter in the decomposition of the wavelet used in DWT.
ilowpassfilter	The low-pass filter in the reconstruction of the wavelet used in DWT.
ihighpassfilter	The high-pass filter in the reconstruction of the wavelet used in DWT.
level	The level of DWT performs.
extensionmode	The extension mode used in DWT.

Examples

Input

The input table, *temperature*, has these columns:

Table 3 - 71: Input table (temperature)

city	date	hour	value
CityA	20121130	0	-1
CityA	20121130	1	-2
CityA	20121130	2	-1
CityA	20121130	3	-1

Table 3 - 71: Input table (temperature) (continued)

city	date	hour	value
CityA	20121130	4	-1
CityA	20121130	5	-2
CityA	20121130	6	-2
CityA	20121130	7	-4
CityA	20121130	8	-5
CityA	20121130	9	-4
CityA	20121130	10	-2
CityA	20121130	11	-1
CityA	20121130	12	-1
CityA	20121130	13	0
CityA	20121130	14	1
CityA	20121130	15	0
CityA	20121130	16	-1
CityA	20121130	17	-2
CityA	20121130	18	-2
CityA	20121130	19	-3
CityA	20121130	20	-2
CityA	20121130	21	-4
CityA	20121130	22	-4
CityA	20121130	23	-4
CityB	20121130	0	12
CityB	20121130	1	12
CityB	20121130	2	11
CityB	20121130	3	10
CityB	20121130	4	9
CityB	20121130	5	9
CityB	20121130	6	9
CityB	20121130	7	9
CityB	20121130	8	10
CityB	20121130	9	12

Table 3 - 71: Input table (temperature) (continued)

city	date	hour	value
CityA	20121130	4	-1
CityA	20121130	5	-2
CityA	20121130	6	-2
CityA	20121130	7	-4
CityA	20121130	8	-5
CityA	20121130	9	-4
CityA	20121130	10	-2
CityA	20121130	11	-1
CityA	20121130	12	-1
CityA	20121130	13	0
CityA	20121130	14	1
CityA	20121130	15	0
CityA	20121130	16	-1
CityA	20121130	17	-2
CityA	20121130	18	-2
CityA	20121130	19	-3
CityA	20121130	20	-2
CityA	20121130	21	-4
CityA	20121130	22	-4
CityA	20121130	23	-4
CityB	20121130	0	12
CityB	20121130	1	12
CityB	20121130	2	11
CityB	20121130	3	10
CityB	20121130	4	9
CityB	20121130	5	9
CityB	20121130	6	9
CityB	20121130	7	9
CityB	20121130	8	10
CityB	20121130	9	12

Table 3 - 71: Input table (temperature) (continued)

city	date	hour	value
CityB	20121130	10	14
CityB	20121130	11	16
CityB	20121130	12	17
CityB	20121130	13	18
CityB	20121130	14	18
CityB	20121130	15	17
CityB	20121130	16	16
CityB	20121130	17	14
CityB	20121130	18	11
CityB	20121130	19	9
CityB	20121130	20	8
CityB	20121130	21	7
CityB	20121130	22	7
CityB	20121130	23	6

SQL-MapReduce Call

```
select * from public.dwt(
    on (select 1) partition by 1
    password('beehive')
    inputtable('public.temperature')
    outputtable('temperature_coef')
    metatable('temperature_meta')
    inputcolumns('value')
    sortcolumn('hour')
    partitioncolumns('city', 'date')
    waveletname('db2')
    level(2)
);
```

Output

Output table (temperature_coef):

Table 3 - 72: Output table (temperature_coef)

city	date	waveletid	waveletcomponent	value
cityA	20121130	1	A2	-2.67075317547306
cityA	20121130	2	A2	-2.75448729810778
cityA	20121130	3	A2	-2.68093013959279

Table 3 - 72: Output table (temperature_coef)

city	date	waveletid	waveletcomponent	value
...

Meta table (temperature_meta):

Table 3 - 73: Meta table (temperature_meta)

city	date	meta	content
cityA	20121130	blocklength	8,8,13
cityA	20121130	length	24
cityA	20121130	waveletname	db2
...

Error Messages

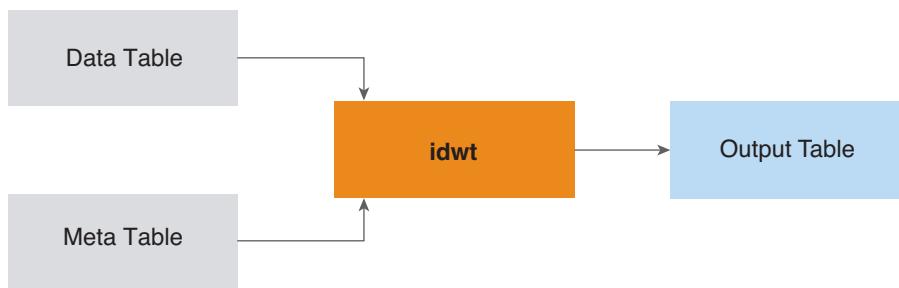
- 1 INPUTTABLE "inputtablename" does not exist
- 2 OUTPUTTABLE "outputtablename" already exists, please remove it
- 3 METATABLE "metatablename" already exists, please remove it
- 4 WAVELETFILTERTABLE "waveletfiltertablename" does not exist
- 5 Either 'WAVELETFILTERTABLE' or 'WAVELETNAME' should be specified to determine wavelet used in transformation.
- 6 Arguments 'WAVELETFILTERTABLE' and 'WAVELETNAME' can not be chosen simultaneously.
- 7 Unrecognized wavelet name.
- 8 Argument clause 'EXTENSIONMODE' should be one of 'sym' 'ppd' or 'zpd'.
- 9 INPUTCOLUMN "inputcolumnname" doesn't exist in inputtable
- 10 PARTITIONCOLUMN "partitioncolumnname" doesn't exist in inputtable
- 11 SORTCOLUMN "sortcolumnname" doesn't exist in inputtable
- 12 Argument clause 'LEVEL' should be a positive integer between 1 and 1000
- 13 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]', start and end should be non-negative integers represent column indices.
- 14 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]' while start < end.
- 15 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]' while start >= 0.

- 16 'INPUTCOLUMNS' requires all input columns to be numeric.
- 17 Column "columnname" has appeared in both 'INPUTCOLUMNS' and 'PARTITIONCOLUMNS' clauses.
- 18 Filter "filterName" is null in WAVELETFILTERTABLE
- 19 Filter "filterName" is duplicate in WAVELETFILTERTABLE
- 20 Filter "filterName" is missed in WAVELETFILTERTABLE
- 21 4 filters in WAVELETFILTERTABLE should be the same length
- 22 Filter "filterName" in WAVELETFILTERTABLE should contain at least two coefficients
- 23 Filter "filterName" in WAVELETFILTERTABLE should consist of numeric values.
- 24 Connection to jdbc:ncluster://"queenip"/"databasename" could not be established.
REASON: JDBC connection could not be made. Test your JDBC connection. It may be caused by wrong arguments in clauses domain, database, userid or password.

IDWT

Summary

IDWT is the inverse function of DWT. IDWT applies inverse wavelet transforms on multiple sequences simultaneously. The inputs are the coefficients of sequences with specified order outputted by DWT and the corresponding wavelet information used in DWT. After running IDWT, the output sequences are the sequences in time domain. Because the output is comparable with the input of DWT, the transformation is also called reconstruction.



Background

These steps describe a typical use case of IDWT:

- 1 Apply DWT to sequences, generate the coefficients of sequences and corresponding meta data.
- 2 Filter the coefficients by various methods (for example, minimum threshold and top n coefficients) according to the object.
- 3 Reconstruct the sequences from the filtered coefficients and compare them with the original ones.

Usage

This syntax assumes that each sequence can be fitted into the memory of the worker.

Syntax (version 1.0)

```
SELECT * FROM idwt(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    METATABLE('meta_table_name')
    OUTPUTTABLE('output_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
    [PARTITIONCOLUMNS('partition_column_name1',
                      'partition_column_name2', ..., 'partition_column_nameN')]
)
;
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, host:port. The host is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [:1]:2406. The port is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	The name of the input table or view contains the coefficients generated by DWT. It should be a table or a view. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
<i>METATABLE</i>	Required	The name of the input table or view contains the meta information used in DWT, it should be a table or a view. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
<i>OUTPUTTABLE</i>	Required	The name of the output table of the reconstruct result. This table must not exist, so if it does exist, you must drop it before running the function again. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
<i>INPUTCOLUMNS</i>	Required	<p>The columns in the input table to be transformed. You can either explicitly list all the names, for example, INPUTCOLUMNS('col1','col2', ...), or specify a range of columns, for example, INPUTCOLUMNS('[4:33]'), or some combination of the above, for example, INPUTCOLUMNS('col1','[4:21]', '[25:53]', 'col73').</p> <p>Ranges are specified using the following syntax: "[<start_column>:<end_column>]", and the column index starts from 0. These columns must contain numeric values between -1e308 and 1e308. If NULL exists in the columns, it is treated as zero.</p>

SORTCOLUMN	Required	The column that represents the order of coefficients in each sequence. The column should be a sequence that consists of serial integer values that start from 1 for each sequence. If some values are missed in the sequence, the corresponding data columns are treated as zero.
PARTITIONCOLUMNS	Optional	One or more columns identify different sequences, every row has the same value in partitioncolumns belong to the same sequence. If not specified, all the rows are treated as one sequence and then a column named "dwt_id**"(* represents random name) filled with value 1 is auto generated in the outputtable as the distribute key. If multiple columns are provided, the first one is adopted as the distribute key of output tables, thus it should be one of the following types: int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea.

All the schema names, table names, and column names are treated as case-insensitive arguments in the function. If either the schema name, the table name, the column name or both the schema and table names include capital letters, you must individually surround each name double-quotation marks.

Input

The input schema of IDWT is the same as the output schema of DWT.

Table 3 - 74: Schema of the input table

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table of DWT)	inherited from input table of DWT	Every row that has the same value in PARTITIONCOLUMNS belongs to the same sequence.
waveletid	int	Index of each wavelet coefficient (starting from 1 for each sequence). If some values are missed in the series, the corresponding input columns are treat as zero.
waveletcomponent	varchar	The component that the coefficient belongs to. Possible values are $A_n D_n D_{n-1} \dots D_1$, where n is the level of the transformation as shown in the background part.
INPUTCOLUMNS (names are inherited from input table)	double	The coefficients after the wavelet transform of the corresponding input column.

Table 3 - 75: Schema of the meta table

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table of DWT)	inherited from input table of DWT	Every row that has the same value in the partition columns belongs to the same sequence.

Table 3 - 75: Schema of the meta table

columns	type	description
meta	varchar	The name of the specified meta information. The possible meta names are listed in Table 3 - 76 .
content	varchar	The value of the corresponding meta name.

For each sequence, the following information is listed in the meta table:

Table 3 - 76: Sequence information

meta	content
blocklength	The length of each component after transformation, from An to D1. For example, 8,8,13.
length	The length of the sequence before transformation. For example, 24.
waveletname	The name of the wavelet used in transformation. For example, db2.
lowpassfilter	The low-pass filter in the decomposition of the wavelet used in DWT.
highpassfilter	The high-pass filter in the decomposition of the wavelet used in DWT.
ilowpassfilter	The low-pass filter in the reconstruction of the wavelet used in DWT.
ihighpassfilter	The high-pass filter in the reconstruction of the wavelet used in DWT.
level	The level of DWT performs.
extensionmode	The extension mode used in DWT.

Output

Returns a message indicating whether the function was successful. The reconstructed sequences are put in the output table. The output table has the following schema:

Table 3 - 77: Output table schema

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	inherited from input table	The values are inherited from input table for each sequence. If the argument PARTITIONCOLUMNS is not specified, a column named "dwt_id**"(* represents a random name), filled with the integer value 1, is used as the partition column.
indexid	int	Index of each reconstructed sequence(start from 1 for each sequence).
INPUTCOLUMNS (names are inherited from input table)	double	Reconstructed data.

Examples

Input

Input table

Table 3 - 78: Input table (temperature_coef)

city	date	waveletid	waveletcomponent	value
cityA	20121130	1	A2	-2.67075317547306
cityA	20121130	2	A2	-2.75448729810778
cityA	20121130	3	A2	-2.68093013959279
...

Meta table

Table 3 - 79: Meta table (temperature_meta)

city	date	meta	content
cityA	20121130	blocklength	8,8,13
cityA	20121130	length	24
cityA	20121130	waveletname	db2
...

SQL-MapReduce Call

```
select * from idwt(
    on (select 1) partition by 1
    password('beehive')
    inputtable('temperature_coef')
    metatable('temperature_meta')
    outputtable('temperature_reconstruct')
    inputcolumns('value')
    sortcolumn('waveletid')
    partitioncolumns('city','date')
) ;
```

Output

Table 3 - 80: Output table (temperature_reconstruct)

city	date	indexid	value
cityA	20121130	1	-1
cityA	20121130	2	-2
cityA	20121130	3	-1
...

Error Messages

- 1 INPUTTABLE "inputtablename" does not exist
- 2 OUTPUTTABLE "outputtablename" already exists, please remove it
- 3 METATABLE "metatablename" does not exist
- 4 INPUTCOLUMN "inputcolumnname" doesn't exist in inputtable
- 5 PARTITIONCOLUMN "partitioncolumnname" doesn't exist in inputtable
- 6 SORTCOLUMN "sortcolumnname" doesn't exist in inputtable
- 7 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]', start and end should be non-negative integers represent column indices.
- 8 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]' while start < end.
- 9 Invalid column range in argument clause 'INPUTCOLUMNS' specified. Range must be of format '[start:end]' while start >= 0.
- 10 'INPUTCOLUMNS' requires all input columns to be numeric.
- 11 Column "columnname" has appeared in both 'INPUTCOLUMNS' and 'PARTITIONCOLUMNS' clauses.
- 12 Meta information "metanames" are missed in meta table.
- 13 Argument clause 'SORTCOLUMN' should be of type integer.
- 14 Connection to jdbc:ncluster://"queenip"/"databasename" could not be established.
REASON: JDBC connection could not be made. Test your JDBC connection. It may be caused by wrong arguments in clauses domain, database, userid or password.

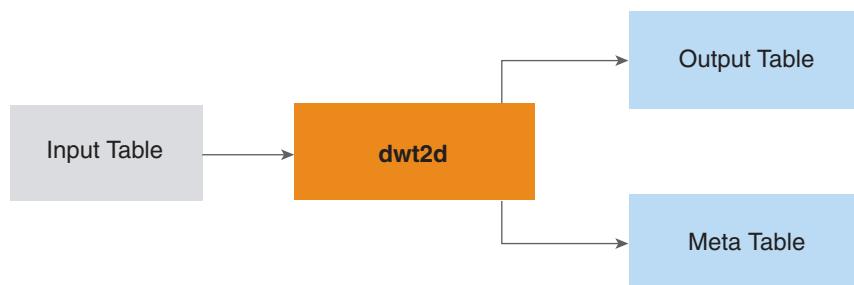
DWT2d

Summary

This function implements wavelet transform on 2 dimensional input, and it is designed to apply wavelet transform on multiple sequences simultaneously. Typically, each sequence is a matrix contains the position in 2 dimensional space and the corresponding values.

Each row in a sequence consists of the sequence identifier (optional), the index of Y axis, the index of X axis, and the corresponding one or more values. Then, you specify the wavelet name, transform level, and optionally the extension mode.

The result is the transformed sequences in Hilbert space with the corresponding component identifier and index. The transformation is also called decomposition as the result is given in multiple components.



A typical use case of DWT2d is:

- 1 Apply DWT2d to the original data and generate the coefficients of matrices and corresponding meta-data.
- 2 Filter the coefficients by various methods (for example, minimum threshold, top n coefficients or just keep the approximate coefficients) according to the object.
- 3 Reconstruct the matrices from the filtered coefficients and compare them with the original ones.

Background

DWT (Discrete Wavelet Transform) is a kind of time-frequency analysis tool for which the wavelets are discretely sampled. Different from the Fourier transform, which provides frequency information on the whole time domain.

A key advantage of DWT is that it provides frequency information at different time points, which is especially useful to keep local information.

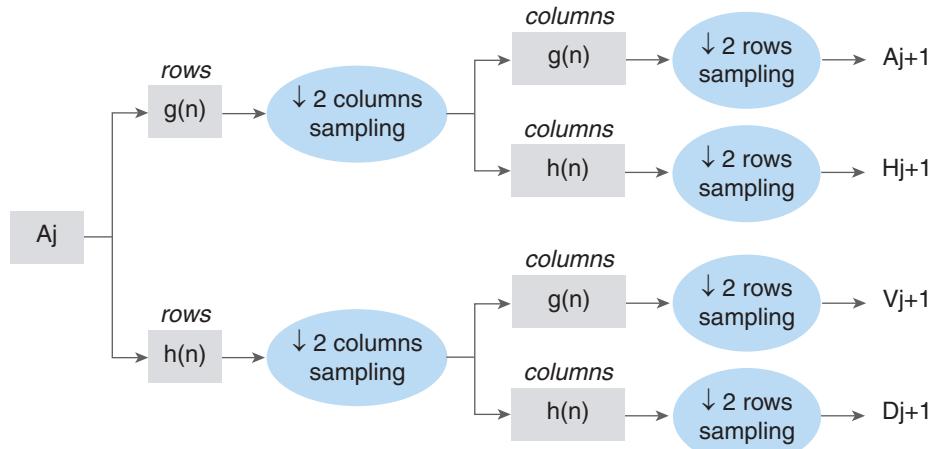
Similar to 1-dimensional DWT, Mallat's algorithm can be represented as the following iterative steps.

- 1 Use the original time domain sequence (2 dimensional matrix) as the input of level 1.
- 2 In level k , each row of the input matrix is convolved with low-pass filter $g(n)$ and high-pass filter $h(n)$, then each convolved rows are downsampled by column, respectively. Thus, two matrices are generated. After that, each column of the two generated matrices are

convolved with the filters $g(n)$ and $h(n)$, and downsampled again. Finally four matrices are given, they are marked as approximation coefficients A_k , horizontal detail coefficients H_k , vertical detail coefficients V_k and diagonal detail coefficients D_k for level n , respectively. The process is shown in [Figure 5](#).

- 3 If current level k reaches max transform level n , then go to the end. Otherwise, A_k is used as the input matrix of next level; increase the current level k by 1, then go to step 2.

[Figure 5: Single level application of dwt2d](#)



$h(n)$ and $g(n)$ represent the high-pass filter and low-pass filter respectively

rows
 $h(n)/g(n)$ Convolve the rows of the input with filter $h(n)$ or $g(n)$

columns
 $h(n)/g(n)$ Convolve the columns of the input with filter $h(n)$ or $g(n)$

Usage

Syntax (version 1.0)

```

SELECT * FROM dwt2d(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    METATABLE('meta_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    [PARTITIONCOLUMNS('partition_column_name1',
        'partition_column_name2', ..., 'partition_column_nameN')]
    INDEXCOLUMNS('indexy_column_name', 'indexx_column_name')
    [RANGE('(starty,startx),(endy,endx)')]
    [WAVELETFNAME('wavelet_name') |
    WAVELETFILTERTABLE('wavelet_filter_table_name') ]
    LEVEL(level)
  )
  
```

```
[EXTENSIONMODE ('extension_mode') ]
[COMPACTOUTPUT ('true' | 'false')]
) ;
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The host is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [:: 1]:2406. The port is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Optional	The name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. Default is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	The name for the input relation. It should be a table or a view. Each row of the relation consists of values of the data and corresponding sequence ID and indexes metric. The input table should contain no more than 1594 columns. If the schema is not specified in the argument, public is automatically added to the argument as the schema name.
<i>OUTPUTTABLE</i>	Required	The name for the output table for the coefficients generated by the wavelet transform. This table must not exist, so if it does exist, you must DROP it before running the function again. If the schema is not specified in this argument, public is automatically added to the argument as the schema name.
<i>METATABLE</i>	Required	The name for the output table for the meta information of the transformation. This table must not exist, so if it does exist, you must drop it before running the function again. If the schema is not specified in the argument, public is automatically added to the argument as the schema name.
<i>INPUTCOLUMNS</i>	Required	The data columns of the input table to be transformed. You can either explicitly list all the names, for example, INPUTCOLUMNS('col1','col2', ...), or specify a range of columns, for example, INPUTCOLUMNS('[4:33]'), or some combination of the above, for example, INPUTCOLUMNS('col1','[4:21]', '[25:53]', 'col73'). Ranges are specified using this syntax: '[start_column:end_column]', and the column index starts from 0. These columns must contain numeric values between -1e308 and 1e308. If NULL exists in the columns, it is treated as zero.
<i>PARTITIONCOLUMNS</i>	Optional	One or more columns identify the different sequences. Every row that has the same value in PARTITIONCOLUMNS belongs to the same sequence. If not specified, all the rows are treated as one sequence and then a column named 'dwt_id**' (the * character represents a random name), filled with value 1, is auto generated in the outputtable and metatable as the distribute key. If multiple columns are provided, the first one is adopted as the distribute key of the output tables and thus, it should have one of these types: int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea.
<i>INDEXCOLUMNS</i>	Required	Specify the columns contain the indexes of input data, two values separated by comma are required for this field. For a matrix, they represent y and x coordinates respectively. The columns should contain integer values. If NULL exists in the columns, the row will be skipped. If all the values are invalid in a sequence, the sequence will be silence ignored.

<i>RANGE</i>	Optional	Specify the start and end coordinates of the input data, integers are required for each value in the field. By default, (<i>starty,startx</i>) is (min value of y, min value of x), (<i>endy,endx</i>) equals to (max value of y, max value of x) in each sequence respectively. If a cell in the range is missed, it is treated as zero. The number of the cells that range specified should be no more than 1,000,000.
<i>WAVELETFILTERTABLE</i>	Optional	The table that stores the coefficients of the filters. The column definitions of the table are listed in Table 3 - 65 . The data read from the table is used in the wavelet transform. If schema is not specified in the argument, public is added to the argument as schema name automatically.
<i>WAVELETNAME</i>	Optional	The name of predefined wavelet filter to be used. The supported predefined wavelet names are listed in Table 3 - 66 .
<i>LEVEL</i>	Required	Identify the level of wavelet transform to be performed. Must be a positive integer in the range of [1,1000].
<i>EXTENSIONMODE</i>	Optional	The method for handling the problem of border distortion. Currently supported values are: 'sym' (replicates boundary value symmetrically), 'zpd' (zeropadding), and 'ppd' (periodic extension). Default is 'sym'. For more information, see Table 3 - 67 .
<i>COMPACTOUTPUT</i>	Optional	Specify whether the rows with all the coefficients values are very small (the absolute value less than 1e-12) should be emitted in output table. If true, those rows will be ignored. This argument is helpful in reducing the size of the output table for sparse matrix. The valid values are "true" and "false", the default value is "true".

You must choose either WAVELETFILTERTABLE or WAVELETNAME to specify the wavelet used in transformation. If both of the parameters are specified, an exception is thrown.

Also, all the schema names, table names, and column names are treated as case-insensitive arguments in the function. If either the schema name, the table name, the column name, or both schema and table names include capital letters, you must individually surround each name with double-quotation marks.

For the more information about WAVELETFILTERTABLE, WAVELETNAME, and EXTENSIONMODE, please refer to DWT.

Input

Table 3 - 81: Input table schema

columns	type	description
PARTITIONCOLUMNS	int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea.	One or more columns identify the different sequences, if PARTITIONCOLUMNS is not provided, a column named "dwt_id**" (* represents random name) filled with value integer 1 is used as the partition column.
indexy	int	Column represents Y-axis index.
indexx	int	Column represents X-axis index.
INPUTCOLUMNS	numeric	One or more columns contain the values for the sequences.

Output

This function generates an output table and a meta table. The output table contains the coefficients and corresponding information, while the meta table contains the global transform information for each sequence.

The schema of the output table is:

Table 3 - 82: Output table schema

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	inherited from input table	The values are inherited from the input table for each sequence. If the argument partitioncolumns is not specified, a column named "dwt_id**" (where * represents a random name) and filled with the value integer 1 is used as partitioncolumns.
waveletid	int	Index of each wavelet coefficient (starts from 1 for each sequence).
waveletcomponent	varchar	The component that the coefficient belongs to, possible values are An Hn Vn Dn Hn-1... H1 V1 D1, where n is the level of the transformation as shown in the background part.
INPUTCOLUMNS (names are inherited from input table)	double	Coefficients after the wavelet transform of the corresponding input column.

The schema of meta table is:

Table 3 - 83: Schema of meta table

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	Inherited from input table	The values are inherited from input table for each sequence, if argument partitioncolumns is not specified, a column named "dwt_id**"(* represents random name) filled with value integer 1 is used as partitioncolumns.
meta	varchar	The identifier of the specified meta information, the possible meta names are listed below.
content	varchar	The value of corresponding meta name.

For each sequence, the following information is listed in the meta table:

Table 3 - 84: Meta table information

PARTITIONCOLUMNS (Sequenceld, name inherited from input)	meta(string)	content(string)
seqid For example, 13341.	blocklength	Pairs that represent the length of each coefficients block (row numbers, column numbers). For example, (5,5), (5,5), (5,6).
seqid For example, 13341.	length	A pair that represents the length of the original sequence in each dimension (row numbers, column numbers). For example, (5,8).
seqid. For example, 13341	range	The min and max indexes of the original sequence. Each index consists of (y-index, x-index). For example, (1,1),(5,8).
seqid For example, 13341.	lowpassfilter	The low-pass filter coefficients used in the decomposition.
seqid For example, 13341.	highpassfilter	The high-pass filter coefficients used in the decomposition.
seqid For example, 13341.	ilowpassfilter	The low-pass filter coefficients used in the reconstruction.
seqid For example, 13341.	ihighpassfilter	The high-pass filter coefficients used in the reconstruction.
seqid For example, 13341.	level	The level of the transform.

Table 3 - 84: Meta table information

PARTITIONCOLUMNS (Sequenceld, name inherited from input)	meta(string)	content(string)
seqid	extensionmode	The extension mode of the transformation. For example, 13341.

Example

Example Input

The sample input of dwt2d is a matrix represented in this table:

Table 3 - 85: Input table

seqid	y	x	value
13453	3	3	10.29524
13453	3	4	10.35803
13453	4	4	9.41787
13453	4	5	9.238792
13454	3	3	-0.47044

Example SQL-MapReduce call

```

SELECT *
FROM dwt2d(
ON (SELECT 1) PARTITION BY 1
PASSWORD ('*****')
INPUTTABLE('wavelet2d')
OUTPUTTABLE('wavelet2d_coef')
METATABLE('wavelet2d_meta')
INPUTCOLUMNS('value')
PARTITIONCOLUMNS('seqid')
INDEXCOLUMNS('y', 'x')
RANGE('(1,1), (5,6)')
WAVELETNAME('db2')
COMPACTOUTPUT('true')
LEVEL(2)
);

```

Example Output

Output Table

Table 3 - 86: wavelet2d_coef

seqid	waveletid	waveletcomponent	value
13453	1	A2	0.0468135620307012
13453	2	A2	0.270299674592108
13453	3	A2	0.0626266069861898
...
13453	84	D1	-4.73265921460517
13454	1	A2	-0.00295439104892103

Please note that some waveletid values are not listed in the table, because the values are almost zero and the argument COMPACTOUTPUT is true.

Meta Table

Table 3 - 87: wavelet2d_meta

seqid	meta	value
13453	blocklength	(3, 3),(3, 3),(3, 3),(3, 3),(4, 4),(4, 4),(4, 4)
13453	length	(5, 6)
13453	waveletname	db2
13453	range	(1,1),(5,6)
...
13454	level	2
...

Error Messages

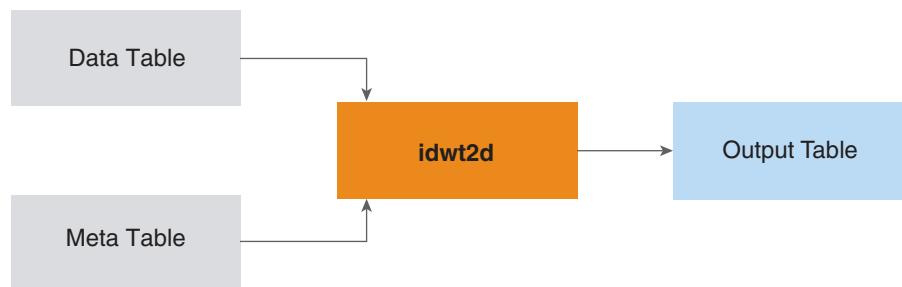
- Connection to jdbc:ncluster://"queenip"/"databasename" could not be established.
REASON: JDBC connection could not be made. Test your JDBC connection. It may be caused by wrong arguments in clauses domain, database, userid or password.
- Unrecognized wavelet name.
REASON: The specified wavelet name is not a supported wavelet name. Refer to [Table 3 - 66, "Supported wavelet names"](#).
- Column "columnname" has appeared in both 'INPUTCOLUMNS' and 'PARTITIONCOLUMNS' clauses.
REASON: A column cannot be used as both INPUTCOLUMN and PARTITIONCOLUMN.

- Either 'WAVELETFILTERTABLE' or 'WAVELETNAME' should be specified to determine wavelet used in transformation.
- Arguments 'WAVELETFILTERTABLE' and 'WAVELETNAME' can not be chosen simultaneously.
- 4 filters in WAVELETFILTERTABLE should be the same length
- Filter "filterName" in WAVELETFILTERTABLE should contain at least two coefficients
- Filter "filterName" in WAVELETFILTERTABLE should consist of numeric values.
- Invalid range for input matrix, number of cells should be no more than 10000000.
REASON: The scale of input matrix is larger than the allowed value.

IDWT2d

Summary

IDWT2d is the inverse function of DWT2d. Or, more specifically, it applies inverse wavelet transforms on multiple sequences simultaneously. The inputs are the coefficients of sequences with specified order outputted by DWT and the corresponding wavelet information generated during the transformation. After running IDWT2d, the output sequences are the original sequences in matrix forms. As the output is comparable with the input of DWT, the transformation is also called reconstruction.



Background

Similar to IDWT, a typical use case of IDWT2d consists of the following steps:

- 1 Apply DWT2d to the original data, generate the coefficients of matrices and corresponding meta data.
- 2 Filter the coefficients by various methods (for example, minimum threshold and top n coefficients) according to the object. Thus we may compress the original matrices.
- 3 Reconstruct the matrices from the filtered coefficients and compare them with the original ones.

Also, there exists other cases. For example, you may find the flaws in the transformed domain according to the energy of different components and map it back to the original position according to the indexes.

Usage

Syntax (version 1.0)

```

SELECT * FROM idwt2d(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    METATABLE('meta_table_name')
    OUTPUTTABLE('output_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
  )
  
```

```
[PARTITIONCOLUMNS ('partition_column_name1',
    'partition_column_name2',...,'partition_column_nameN')
[COMACTOUTPUT('true' | 'false')]
) ;
```

Arguments

DOMAIN	Optional	Has the form, host:port. The host is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [:: 1]:2406. The port is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
DATABASE	Optional	This is the name of the database where the input table is present. Default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
PASSWORD	Required	The Aster Database password of the user.
INPUTTABLE	Required	The name of the input table or view contains the coefficients generated by DWT2D. It should be a table or a view. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
METATABLE	Required	The name of the input table or view contains the meta information used in DWT2D, it should be a table or a view. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
OUTPUTTABLE	Required	The name of the output table of the reconstruct result. This table must not exist, so if it does exist, you must drop it or choose a different name before running the function again. If the schema is not specified in the argument, "public" is automatically added to the argument as the schema name.
INPUTCOLUMNS	Required	The columns in the input table to be transformed. You can either explicitly list all the names, for example, INPUTCOLUMNS('col1','col2', ...), or specify a range of columns, for example, INPUTCOLUMNS('[4:33]'), or some combination of the above, for example, INPUTCOLUMNS('col1','[4:21]', '[25:53]', 'col73'). Ranges are specified using the following syntax: "[start_column:end_column]", and the column index starts from 0. These columns must contain numeric values between -1e308 and 1e308. If NULL exists in the columns, it is treated as zero.
SORTCOLUMN	Required	The column that represents the order of coefficients in each sequence. The column should be a sequence that consists of serial integer values that start from 1 for each sequence. If some values are missed in the sequence, the corresponding data columns are treated as zero.
PARTITIONCOLUMNS	Optional	One or more columns identify different sequences, every row has the same value in partitioncolumns belong to the same sequence. If not specified, all the rows are treated as one sequence and then a column named "dwt_id**"(* represents random name) filled with value 1 is auto generated in the outputtable as the distribute key. If multiple columns are provided, the first one is adopted as the distribute key of output tables, thus it should be one of the following types: int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea.
COMPACTOUTPUT	Optional	Specify whether the rows with all the coefficients values are very small (the absolute value less than 1e-12) should be emitted in output table. If true, those rows will be ignored. This argument is helpful in reducing the size of the output table for sparse matrix. The valid values are "true" and "false", the default value is "true".

All the schema names, table names, and column names are treated as case-insensitive arguments in the function. If either the schema name, the table name, the column name or both the schema and table names include capital letters, you must individually surround each name double-quotation marks.

Input

The input schema of IDWT2d is the same as the output schema of DWT2d. The schema of the input table is described in [Table 3 - 82](#).

The schema of meta table is described in [Table 3 - 83](#).

Output

Returns a message indicating whether the function was successful. The reconstructed matrices are put in the output table. The output table has the following schema:

Table 3 - 88: Output table schema

columns	type	description
PARTITIONCOLUMNS (names are inherited from input table)	int, smallint, bigint, numeric, numeric(p), numeric(p,a), text, varchar, varchar(n), UUID, or bytea. inherited from input table	One or more columns identify the different sequences. If the argument PARTITIONCOLUMNS is not specified, a column named "dwt_id**"(* represents a random name), filled with the integer value 1, is used as the partition column.
indexy	int	column represents Y-axis index for reconstructed matrices
indexx	int	column represents X-axis index for reconstructed matrices
INPUTCOLUMNS	numeric	one or more columns contain the values for the sequences

Example

As the inverse function of DWT2d, it is adopted to reconstruct the original matrix from the coefficients with the meta data.

Example Input

Input Table

Table 3 - 89: wavelet2d_coef

seqid	waveletid	waveletcomponent	value
13453	1	A2	0.0468135620307012
13453	2	A2	0.270299674592108
13453	3	A2	0.0626266069861898
...
13453	84	D1	-4.73265921460517
13454	1	A2	-0.00295439104892103

Meta Table

Table 3 - 90: wavelet2d_meta

seqid	meta	value
13453	blocklength	(3, 3),(3, 3),(3, 3),(3, 3),(4, 4),(4, 4),(4, 4)
13453	length	(5, 6)
13453	waveletname	db2
13453	range	(1,1),(5,6)
...
13454	level	2
...

Example SQL-MapReduce call

```
SELECT * FROM idwt2d(
    ON (SELECT 1) PARTITION BY 1
    password('*****')
    INPUTTABLE('wavelet2d_coef')
    METATABLERE ('wavelet2d_meta')
    OUTPUTTABLE('wavelet2d_reconstruct')
    INPUTCOLUMNS ('value')
    SORTCOLUMN ('waveletid')
    PARTITIONCOLUMNS ('seqid')
) ;
```

Output Example

Table 3 - 91: wavelet2d_reconstruct

seqid	indexy	indexx	value
13453	3	3	10.29524
13453	3	4	10.35803
13453	4	4	9.41787000000001
13453	4	5	9.23879200000001
13454	3	3	-0.47044

Because the argument COMPACTOUTPUT is not specified, it takes the default value, which is 'true'. In this case, the other zero values in the range (1,1),(5,6) are ignored in the output table.

Error Messages

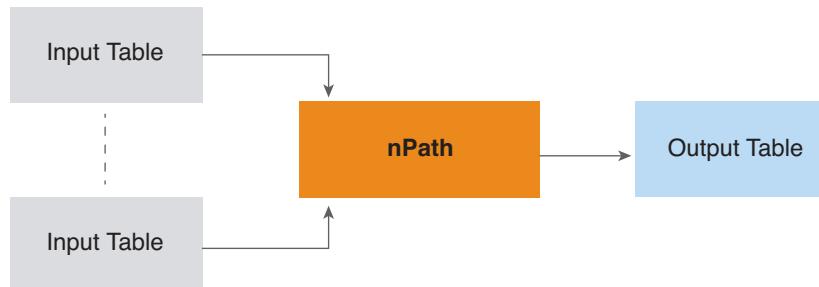
- ERROR: Connection to jdbc:ncluster://"queenip"/"databasename" could not be established.
REASON: The JDBC connection could not be established. Test your JDBC connection. This may be caused by wrong information supplied in the DOMAIN, DATABASE, USERID or PASSWORD arguments.
- ERROR: Column "columnname" has appeared in both 'INPUTCOLUMNS' and 'PARTITIONCOLUMNS' clauses.
REASON: A column cannot be specified as both INPUTCOLUMN and PARTITIONCOLUMN.
- ERROR: Meta information "metanames" are missed in meta table.
REASON: Some rows are missing or corrupted in the meta table. Check the meta table.

CHAPTER 4 Pattern Matching with Teradata Aster nPath

- What is Teradata Aster nPath?
- nPath
- Patterns, Symbols, and Operators in Teradata Aster nPath
- Teradata Aster nPath Examples

What is Teradata Aster nPath?

The Teradata Aster nPath SQL-MR function allows you to perform regular pattern matching over a sequence of rows from one or more inputs. For clarity, this document refers to each sequence of matched rows as a *matched pattern*.



With Teradata Aster nPath, you can find sequences of rows that match a pattern of your choice. The Teradata Aster nPath function lets you use symbols when building patterns. Symbols let you define the pattern matching conditions and help you extract information from these matched rows.

The Teradata Aster nPath function lets you:

- use a regular expression to specify a pattern you want to match in an ordered collection of rows and label individual matching rows with symbols; and
- compute SQL aggregates on or find particular values in each matched pattern (Teradata Aster nPath uses the symbols you define for matching rows to get these aggregates and values).

The Teradata Aster nPath function uses regular expressions because they are simple, widely understood, and flexible enough to express most search criteria. While most uses of regular expressions focus on matching patterns in strings of text; Teradata Aster nPath enables matching patterns in *sequences of rows*.

The Teradata Aster nPath function performs fast analysis on ordered data. The clauses in Teradata Aster nPath let you express complicated pathing queries and ordering relationships that might otherwise require you to write multi-level joins of relations in SQL. With Teradata Aster nPath, you indicate a desired ordering and then specify a pattern that is matched across the ordered rows of data.

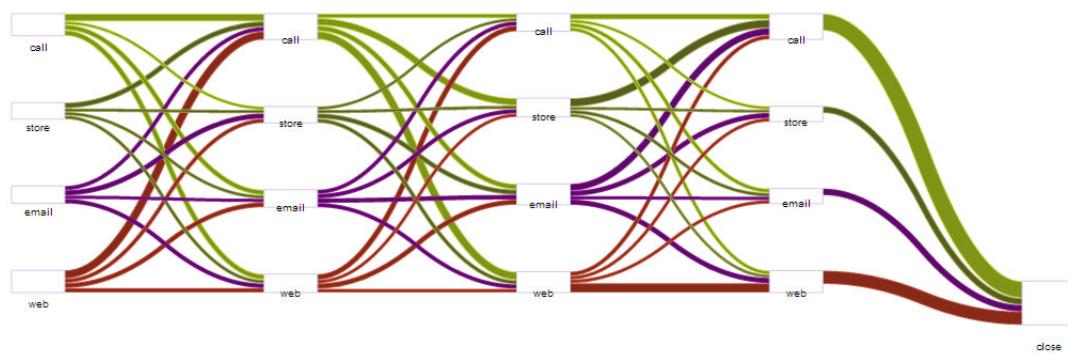
For each matched pattern in the sequence of rows, Teradata Aster nPath generates a row of output that contains SQL aggregates computed over the rows in the matched pattern.

Applications of Teradata Aster nPath

In its simplest application, you can use Teradata Aster nPath to compute SQL 1999 aggregates such as RANK, LAG/LEAD, running aggregates, FIRST_VALUE, LAST_VALUE, and others. Many applications of Teradata Aster nPath can compute aggregates over sequences that cannot be expressed in SQL 1999, or that would require self-joins for multiple passes over the data if expressed in SQL 1999.

Additionally, the output of Teradata Aster nPath is particularly suited for generating informative graphs using third-party data graph generators.

Figure 6: Example of a Sankey Diagram of Teradata Aster nPath Output



Here are more examples of what you can do with Teradata Aster nPath:

- Weeding out irrelevant data from a dataset so it becomes more meaningful. Then feed the results into other functions.
- Categorizing entities based on the patterns observed. For example, “loyal customers” or “price sensitive shoppers.”

Multiple Input Support

The Teradata Aster nPath function supports multiple inputs, which provides these advantages:

- **Avoids casting:** When merging data from multiple input streams, it is not necessary to cast to a common data type.
- **Eliminates joins:** If the inputs contain dimensional look-up tables, you don't have to join the main table with these small dimensional tables. By using a dimensional input, you effectively create a duplicate copy of the dimensional table for every row on which the function will operate.
- **Eliminates unions:** If you want to merge multiple fact tables, the individual tables are sorted and merged within Teradata Aster nPath. Without multiple inputs, you would have to perform a union, which causes a scan and sort on a larger union-ed table.

For more information about SQL-MR with multiple inputs, see “Introducing SQL-MR with Multiple Inputs” in the *Aster Database User Guide*.

nPath

Permissions

Before running the Teradata Aster nPath function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “GRANT EXECUTE on a function” in the Aster Database User Guide.

Syntax (version 1.0)

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
       * | expression [ [ AS ] output_name ] [, ...]
FROM NPATH
( on_clause1 [ on_clause2 ... on_clausen ]
  MODE ( OVERLAPPING | NONOVERLAPPING )
  PATTERN ( 'pattern_of_symbols' )
  SYMBOLS ( symbol_predicate AS symbol [, ...] )
  RESULT ( aggregate_function( expression OF symbol ) AS alias [, ...] )
)
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ]
  [, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ];
```



Tip: You can write the clauses in any order.

ON Clause

The Teradata Aster nPath function can work on multiple input streams, with each one defined in its own ON clause. The ON clause specifies the input relation, which can be a table, view, or query. If you provide a query, you must enclose it in parentheses. Note that Teradata Aster nPath does not accept PARTITION BY ANY inputs.

The syntax of an *on_clause* is:

```
partition_attributes_input | dimensional_input
```

PARTITION BY Expression

The PARTITION BY expression defines the scope of a partition of input rows over which Teradata Aster nPath searches for pattern matches. If there are multiple PARTITION BY expressions, they must use the same partition columns. The inputs are then cogrouped using the specified columns before Teradata Aster nPath acts on them. See “How a SQL-MR function performs a cogroup” in the Aster Database User Guide.

The syntax of *partition_attributes_input* is:

```
table_input PARTITION BY partitioning_attributes [order_by]
```

DIMENSION Expression

The DIMENSION expression defines a set of input rows that Aster Database makes available to Teradata Aster nPath for *every* partition. See “How dimensional inputs work in SQL-MR” in the Aster Database User Guide.

The syntax of *dimensional_input* is:

```
table_input DIMENSION [order_by]
```

where *table_input* is:

```
ON table_expression [ AS alias ]
```

where *table_expression* is:

```
{ table_name | view_name | ( query ) }
```

ORDER BY Expression

The ORDER BY expression specifies the sort order of the input rows.

MODE Clause

The MODE clause indicates whether matched PATTERNs may overlap. After finding one sequence of rows that matches the specified pattern, Teradata Aster nPath looks for the next match. To begin the next pattern search, the choice of the starting row depends on the match mode you have chosen:

- In OVERLAPPING match mode, Teradata Aster nPath finds every occurrence of the pattern in the partition, regardless of whether it might have been part of a previously found match. This means that, in OVERLAPPING mode, one row can match multiple symbols in a given matched PATTERN.
- In NONOVERLAPPING match mode, Teradata Aster nPath begins the next pattern search at the row that follows the last PATTERN match. Note that this is the default behavior of

many commonly used pattern matching utilities like the popular *grep* utility in UNIX systems.

PATTERN Clause

The PATTERN clause defines the pattern that Teradata Aster nPath searches for.

Symbols and Operators

You express the PATTERN using symbols and operators. For example, to match every instance in which a row that matches symbol A is followed directly by a row that matches symbol B, use the pattern “A.B” (the dot operator means “is followed by”). See “[Patterns, Symbols, and Operators in Teradata Aster nPath](#)” on page 162.

If a particular sub-sequence has to appear multiple times within the pattern, you can use the range-matching feature in Teradata Aster nPath. See “[Matching Repeated Patterns in Teradata Aster nPath](#)” on page 165.

SYMBOLS Clause

The SYMBOLS clause defines the row elements in the pattern, expressed as a comma-separated list of symbol definitions. Each symbol definition has this form:

symbol_predicate AS symbol

In this form:

- *symbol_predicate* is an SQL predicate.
- *symbol* is a case-insensitive string that represents the rows that match this predicate.

It is common to define each symbol as just one or two uppercase letters because short symbols are easy to read when assembled into a pattern expression.

For example, a SYMBOLS clause for analyzing website visits might look like this:

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

Symbol Qualification in Multiple Input Cases

When using multiple inputs, each symbol must come from one and only one input stream. To resolve ambiguity (if necessary), you can qualify the column reference in the SYMBOLS clause with the table name. This is only necessary if the input tables contain columns that have the same name. This example uses a SYMBOLS clause that references columns from two tables (weblog table and ads table):

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

Applying a Symbol to a Row

The Teradata Aster nPath function applies a symbol to a row only if the row satisfies the *symbol_predicate* part of the symbol definition.

For example, consider this symbol definition:

```
pagetype = 'homepage' AS H
```

This symbol definition applies to the first and fourth body rows of this table:

Table 4 - 92: Input Table Example

sessionid	clicktime	userid	productname	pagetype	referrer	productprice
1	07:00:10	333		home	www.yahoo.com	
1	07:00:12	333	ipod	checkout	www.yahoo.com	200.2
1	07:01:00	333	bose	checkout		340
13	15:35:08	67403		home	www.google.com	

If the Teradata Aster nPath function encounters a null value when trying to match the symbol predicate, Teradata Aster nPath considers this occurrence a non-match.

A symbol may be associated with the predicate “true”, meaning that the symbol can match any row. For example, TRUE AS A.



Tip: The predicates of different symbol definitions can overlap, and therefore multiple symbols may match the same row.

Comparing the Current Row to a Preceding Row

In your symbol predicate, you can compare the current row to a *preceding row* to determine whether it is considered a match for the symbol. See [“LAG expressions in symbol predicates” on page 166](#).

RESULT Clause: Teradata Aster nPath Output

The RESULT clause defines the output columns of the Teradata Aster nPath function as a comma-separated list of expressions. The Teradata Aster nPath function evaluates the RESULT clause once for every matched pattern in the partition. In other words, Teradata Aster nPath generates one output row per pattern match.

In the RESULT clause, each *expression* operates on one or more *symbols*, and each expression is followed by the *alias* to be applied to this column of output. The form of an output column definition in the RESULTS clause is:

```
aggregate_function ( expression OF symbol ) AS alias
```

Since each *symbol* represents all the rows that matched that symbol’s predicate in this particular matched pattern, you must specify an *expression* (often just the column name) to state what values you want to retrieve from the matched rows, and then apply an *aggregate*

function to the results of that *expression*, in order to generate a single, useful value from the set of matched rows in the *symbol*.

For example, imagine that we want to count how many product pages a web visitor viewed during a visit to our website. To do this, the output column definition in the RESULT clause might look like the following. In this example, the symbol *PP* represents the rows that record a user's views of the *product pages* on a website.

```
COUNT ( * OF PP ) AS count_product_pages_visited
```

For a list of supported aggregate functions, see “[Results: Applying an SQL Aggregate](#)” on [page 170](#).

Working with Teradata Aster nPath Output

The output rows from Teradata Aster nPath can subsequently be used like the results of any SQL query. Rows from Teradata Aster nPath may be filtered outside of it using WHERE, aggregated using GROUP BY (with groups optionally vetted using a HAVING clause), sorted using ORDER BY, de-duplicated using DISTINCT/DISTINCT ON, truncated using LIMIT/OFFSET, and so on.

Patterns, Symbols, and Operators in Teradata Aster nPath

The Teradata Aster nPath function performs pattern matching and returns a row with aggregates for each matched pattern. This section describes what a pattern is, what it means to match a pattern against a sequence of rows, and how each matched pattern translates to an output row.

Note that the SYMBOLS and PATTERN arguments do not have any order associated with them. You can write the clauses in the order you want and system reads them in the order it needs to.

Patterns

A pattern consists of several elements: symbols, operators, nesting parentheses, and anchors. Below is a simple pattern that matches every instance in which a row of type *B* follows a row of type *A*. In this example, *A* and *B* are symbols, and the dot is the operator:

A.B

You can write a PATTERN definition so that it matches only those patterns that *repeat*, or that contain repeated elements. See “[Matching Repeated Patterns in Teradata Aster nPath](#)” on [page 165](#).

Pattern Matching

Conceptually, the pattern matching of Teradata Aster nPath proceeds like this: Starting from a row in a partition, Teradata Aster nPath tries to match the given pattern along the row sequence in the partition (recall that the rows within a partition are ordered as specified in the ORDER BY clause).

If a match is not possible starting at the current row, nothing is outputted. Otherwise, Teradata Aster nPath continues to the next row. When Teradata Aster nPath finds a sequence of rows that match the PATTERN, it picks the largest set of rows that constitute that match and generates an output row based on this match, as discussed next.

Consider a match starting at a row t_1 and ending at the row t_4 . For this example, let's assume the pattern to be matched is ' $A.B^+$ '. In this case, let's assume further that t_1 maps to the symbol A , and each row from t_2 through t_4 maps to the symbol B . (This means that each of the rows t_2 , t_3 , and t_4 individually satisfies the symbol predicate for B .) After the matching is complete, our symbol A represents row t_1 , and symbol B represents rows t_2 , t_3 , and t_4 in the matched PATTERN.

Now that the symbols are populated with rows from the match, Teradata Aster nPath evaluates the RESULT clause to generate output using the data in the symbols. Typically, this amounts to applying an SQL aggregate to each symbol for the match. Teradata Aster nPath returns one row with the result values, and proceeds to search for the next PATTERN match.

When creating patterns in practice, it helps to create a simple set of data in which the pattern you are searching for is included. Create your Teradata Aster nPath function call and run it against the known small data set, to ensure the data you expected is selected. Refine the PATTERN until the outcome is as you want it, and then run your function call over the larger set of data.

Greedy Pattern Matching

Pattern matching in nPath is greedy, but the operators are non-greedy (also known as lazy). The nPath function always finds the longest available match no matter what operators are present in the pattern.

For example, consider this dataset:

Table 4 - 93: Example 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

Also, consider this nPath pattern matching query:

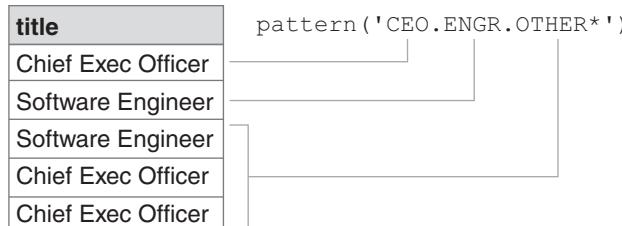
```
select job_transition_path, count(*) as count
from npath(on link2
partition by userid order by startdate
mode(nonoverlapping)
pattern('CEO.ENGR.OTHER*')
symbols(title ilike 'software eng%' as ENGR, true as OTHER,
       title ilike 'Chief Exec Officer' as CEO)
result(accumulate(title of ANY(ENGR, OTHER, CEO)) as job_transition_path))
group by 1 order by 2 desc;
```

This query returns one match (which contains 5 rows), as shown in the example output table:

Table 4 - 94: Example output table

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

In the **CEO.ENGR.OTHER*** pattern, **CEO** matches the first row, **ENGR** matches the second row, and **OTHER*** matches the remaining rows, as shown below.



Now consider this query:

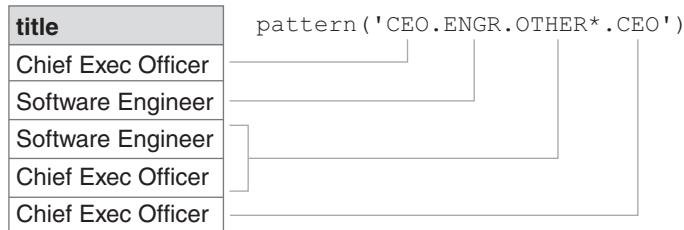
```
select job_transition_path, count(*) as count
from npath(on link2
partition by userid order by startdate
mode(nonoverlapping)
pattern('CEO.ENGR.OTHER*.CEO')
symbols(title ilike 'software eng%' as ENGR, true as OTHER,
       title ilike 'Chief Exec Officer' as CEO)
result(accumulate(title of ANY(ENGR, OTHER, CEO)) as job_transition_path))
group by 1 order by 2 desc;
```

This query, just like the previous query, returns one match (which contains 5 rows), as shown in the example output table:

Table 4 - 95: Example output table

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

In the **CEO.ENG.R.OTHER*.CEO** pattern, **CEO** matches the first row, **ENG.R.** matches the second row, **OTHER*** matches the next two rows, and **CEO** matches the last row, as shown below.



Matching Repeated Patterns in Teradata Aster nPath

If a particular sub-sequence has to appear multiple times within the PATTERN, you can easily represent this requirement using the *range matching* feature in Teradata Aster nPath. (Here, we use the term “sub-sequence” to refer to any portion of the PATTERN that you enclose in parentheses.) The range matching feature allows you to specify the minimum and maximum number of times a sub-sequence must appear in the sequence. The repetition count thresholds for sub-sequences can be specified in one of the following formats:

- *sub-sequence{n}*

The {n} range specifies that the sub-sequence must appear exactly n number of times to be considered a match.

- *sub-sequence{n,}*

The {n,} range specifies that the sub-sequence must appear at least n number of times to be considered a match.

- *sub-sequence{n,m}*

The {n,m} range specifies that the sub-sequence must appear at least an n number of times and at most m number of times to be considered a match.

For example, if the sub-sequence (A.B|C) should appear exactly three times in the sequence, then you can represent the pattern in the following way using the PATTERN clause:

```
PATTERN('X.(Y.Z).(A.B|C){3}')
```

This is equivalent to the pattern

```
PATTERN('X.(Y.Z).(A.B|C).(A.B|C).(A.B|C)')
```

If the sub-sequence (A.B|C) should appear at least four times, you can represent the pattern in the following way:

```
PATTERN('X.(Y.Z).(A.B|C){4,}')
```

which is same as

```
PATTERN('X.(Y.Z).(A.B|C).(A.B|C).(A.B|C).(A.B|C).*(A.B|C)*)')
```

If the sub-sequence (A.B|C) should appear at least two times and at most four times, you can represent the pattern in the following way:

```
PATTERN('X.(Y.Z).(A.B|C){2,4}')
```

which is same as

```
PATTERN('X. (Y.Z) . (A.B|C) . (A.B|C) . (A.B|C) ? (A.B|C) ?')
```

Symbols

The meaning of a symbol depends on its context:

- In a *PATTERN clause*, a symbol represents a row of a particular type that you're searching for as part of a row sequence. (In the *SYMBOLS* clause, you write a predicate to define the type of row that matches the symbol.)
- In the *RESULT clause*, a symbol represents all the rows that matched that symbol's predicate in this particular matched *PATTERN*. This allows you to apply an aggregate function to all the symbol's rows in the matched *PATTERN*.

Teradata Aster nPath uses any valid identifier (a character, followed by characters and digits) as a symbol. Symbols are *not* case sensitive; for example *A* and *a* refer to the same symbol. It's common to use short symbols of one or two letters to make your patterns easier to read.

Each symbol is the result of a match of a *symbol predicate* you defined in your *SYMBOLS* clause (see “[Syntax \(version 1.0\)](#)” on page 158 for instructions). Here's a simple symbol predicate that defines a symbol, *H*, to match any row in which a column “pagetype” contains the value “homepage”:

```
pagetype = 'homepage' AS H
```

LAG expressions in symbol predicates

You can compare the current row with previously seen rows to decide if a symbol applies to it. To do this, use the LAG expression in your *SYMBOLS* clause. You write the LAG expression in either of the following formats:

```
LAG ( expression-prev, lag-rows [, default-value ] )
      operator expression-current
```

or

```
expression-current operator LAG
( expression-prev, lag-rows, [ default_value ] )
```

where:

- *expression-current* is the name of the column from the current row, or an expression operating on this column.
- the *operator* can be *>*, *>=*, *<*, *<=*, *=*, or *!=*
- *expression-prev* is the name of the column from the current row, or an expression operating on this column.
- *lag-rows* is the number of rows to count back from the current row to reach the row we designate as the *earlier row*. For example, to compare with the immediately preceding row, use “1”.
- *default-value* is the optional value to be used when there are no rows which can be designated as “earlier row,” in such a case the *default-value* will be evaluated on the current row and used in place of the *expression-prev*.

To evaluate the LAG expression, Aster Database uses the *operator* to compare the value of *expression-current* with the value of *column-previous*.

If you use a LAG expression in a SYMBOLS clause and the input is not a table, you must create an alias of the input query as show in the example below; otherwise, an error is thrown.

```
SELECT * FROM NPATH
(
    on (select customer_id, session_id,
          datestamp,page from bank_web_clicks) ALIAS
    PARTITION BY customer_id, session_id
    ORDER BY datestamp
    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)
)
```

Rules for Using LAG Expressions

- You can use multiple LAG expressions to define a symbol.
- If your symbol definition includes a LAG expression, the definition *cannot* contain a disjunction (OR operator).



Tip: When using the LAG() function to define a symbol, the LAG() function can either be on the left-hand side (LHS), the portion to the left of the relational operator, or the right-hand side (RHS) of an expression, as in these two examples, which check that the time interval between the last row and the current row is less than one hour:

```
rec_date - Lag(rec_date,1) < '1 hour'::Interval AS B
Lag(rec_date,1) > rec_date - '1 hour'::Interval AS B
```

Lag expression example

The example below uses a LAG expression in a symbol definition. Here, for all the patterns where the user visits the home page, then visits checkout pages and buys increasingly expensive products (in sequence), this Teradata Aster nPath query will find the first product bought and the most expensive product bought.

```
SELECT *
  FROM NPATH
  (
    ON aggregate_clicks
    PARTITION BY sessionid
    ORDER BY clicktime ASC
    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::REAL ) AS P2
)
RESULT
(
    FIRST ( productprice OF P1 ) AS first_product,
    MAX_CHOOSE ( productprice, productname OF P2 ) AS max_product,
    FIRST ( sessionid OF P2 ) AS sessionid
)
)
ORDER BY sessionid ;
```

Operators

You can form patterns by combining symbols with these operators:

Table 4 - 96: Teradata Aster nPath Operators

Operator	Meaning
.	is followed by; the cascade operator. The cascade operator separates the first symbol from the next symbol in the order of occurrence. The expression $A.B$ means “A is followed by B”. (Note! The dot operator does not represent a wildcard as in some other regular expression syntaxes.) Also used for occurs exactly one time.
	or; the alternative operator. The alternative operator occurs between two or more alternatives.
?	occurs at most once. This is a frequency operator, and should come directly after the symbol upon which it operates.
*	occurs zero or more times. This is a frequency operator, and should come directly after the symbol upon which it operates.
+	occurs at least once. This is a frequency operator, and should come directly after the symbol upon which it operates. If you want to match the case where a symbol (or pattern) repeats a <i>specific number of times</i> , use the range matching feature as explained in “ Matching Repeated Patterns in Teradata Aster nPath ” on page 165 .
	To specify occurs exactly one time, use no operator or the cascade operator (.)
^	occurs at the beginning of the pattern. This operator should come directly before the symbol upon which it operates.
\$	occurs at the end of the pattern. This operator should come directly after the symbol upon which it operates.



Tip: A symbol without any operator (other than the cascade operator (“.”)) represents exactly one occurrence. Also, the ?, *, and + operators are non-greedy.

The precedence of operators is, from highest to lowest:

- 1 Frequency operators (“?”, “*”, “+”)
- 2 Cascade operator (“.”)
- 3 Alternative operator (“|”)

Operators with equal precedence associate left to right.

The following examples show the correct precedence that will be used for some simple Teradata Aster nPath patterns:

- $A.B+$ equals $A.(B+)$
- $A|B^*$ equals $A|(B^*)$
- $A.B|C$ equals $(A.B)|C$

Nesting parentheses

Patterns can be nested using parentheses “(” and “)”.

Anchors

The special characters “^” and “\$” are placeholders for the start and the end of the sequence being searched, respectively. The character “^” only makes sense at the start of a pattern, and “\$” only makes sense at the end of a pattern.

For example, to search the field named “City” for cities whose names start with “Altos” (for example, “Altos Oaks”), you can use this regular expression: “`^Altos`”. This expression matches “Altos Oaks” but not “Los Altos.”



In the context of Teradata Aster nPath, the special characters “^” and “\$” refer to the first and last rows in each partition defined by the PARTITION BY clause.

For example, suppose that a table stores information about all cities where packages are picked up or dropped off by delivery trucks. The table records the truck ID, the city name, and the time (timestamp) of when the truck departed that city. To find all trucks whose journeys started in “Altos Oaks,” but not all trucks whose journeys passed through it at some point, then the SYMBOL and PATTERN clauses would look something like the following:

```
PARTITION BY truckID
ORDER BY departureTimestamp
SYMBOLS (CityName = "Altos Oaks" AS a ...)
PATTERN ('^a')
```

In this example, note the importance of the PARTITION BY and ORDER BY clauses.

Partitioning by truck ID allows the query to look at each truck independently. Ordering by truck departure time allows the query to find each truck's original departure point. In order to find all trucks whose journeys ended in “Altos Oaks” the PATTERN would be ‘a\$’.



In a more realistic example, you may want to partition by truck ID and date because a truck starts a new journey on a daily, weekly, or some other time interval basis.

Results: Applying an SQL Aggregate

In the Teradata Aster nPath RESULT clause, you compute aggregates over each matched pattern.

Supported SQL aggregates include: AVG, COUNT, MAX, MIN, and SUM.

The special Teradata Aster nPath sequence aggregates:

`FIRST`

`FIRST (pageid OF B)`—Gives the pageid (here we use “pageid” as an example column name) of the first row in the match that maps to `B` (row `t2` in our example in “[Pattern Matching](#)” on page 163).

`LAST`

`LAST (pageid OF B)`—Gives the pageid of the last row that maps to symbol `B` in the match.

FIRST_NOTNULL	<code>FIRST_NOTNULL (pageid OF B)</code> —Gives the first non-null pageid among the rows that map to <i>B</i> .
LAST_NOTNULL	<code>LAST_NOTNULL (pageid OF B)</code> —Gives the last non-null pageid among the rows that map to <i>B</i> .
MAX_CHOOSE	<code>MAX_CHOOSE (product_price, product_name OF B)</code> —Gives the <code>product_name</code> of the most expensive product among the rows that map to <i>B</i> . The <code>MAX_CHOOSE</code> function takes the form, <code>MAX_CHOOSE (quantifying_column, descriptive_column OF symbol)</code> and returns the <code>descriptive_column</code> value of the row with the highest-sorted <code>quantifying_column</code> value. The <code>qualifying_column</code> has a sortable datatype (smallint, integer, biginteger, real, date, time, timestamp, varchar, and character are supported) and the <code>descriptive_column</code> can be of any datatype.
MIN_CHOOSE	<code>MIN_CHOOSE (product_price, product_name OF B)</code> —Gives the <code>product_name</code> of the least expensive product among the rows that map to <i>B</i> . The <code>MIN_CHOOSE</code> function operates like <code>MAX_CHOOSE</code> , but returns the <code>descriptive_column</code> value of the lowest-sorted row. It supports the same argument datatypes as <code>MAX_CHOOSE</code> .
DUPCOUNT	<code>DUPCOUNT (<expression> OF ANY(<symbol list>))</code> —For each row in the row sequence for the symbol list, this aggregate counts the number of times the current value of <code><expression></code> has appeared immediately preceding this row. When <code><expression></code> is also the ORDER BY expression, this is equivalent to <code>ROW_NUMBER() - RANK()</code> .
DUPCOUNTCUM	<code>DUPCOUNTCUM (<expression> OF ANY(<symbol list>))</code> —For each row in the row sequence for the symbol list, the number of duplicate values of <code><expression></code> that have appeared contiguously preceding this row. When <code><expression></code> is also the ORDER BY expression, this is equivalent to <code>ROW_NUMBER() - DENSE_RANK()</code> .
ACCUMULATE	<code>ACCUMULATE (pageid of B)</code> concatenates the <code>pageid</code> column for every row that matches symbol <i>B</i> .

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*.

Example

This example shows the use of `LAST_NOTNULL`, `MAX_CHOOSE`, and `MIN_CHOOSE`.

Input Table

Table 4 - 97: Input Table

userid	gender	ts	productname	productamt
1	M	1/1/2012	Cubic	1
1	M	2/1/2012	Octagon	10
1	M	3/1/2012	Pentagon	100
1	M	4/1/2012	Pyramid	1000
2		1/1/2012	Rectangle	2
2		2/1/2012	Sphere	20
2	F	3/1/2012	Globe	200
3	F	1/1/2012	Spire	3
3	F	2/1/2012	Trapezoid	30
3	F	3/1/2012	Triangle	300

Syntax

```
SELECT *
FROM NPATH(
    ON trans1
    PARTITION by userid order by ts
    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)
) ;
```

Output

Table 4 - 98: Output Table

Userid	gender	max_prod	min_prod
1	M	Pyramid	Cubic
2	F	Globe	Rectangle
3	F	Triangle	Spire

Teradata Aster nPath Examples

- [Clickstream Data](#)
- [Lead](#)
- [Rank](#)
- [Complex Path Query](#)
- [Multiple Inputs with Partitioned Inputs](#)
- [Multiple Inputs with Dimensional Input](#)
- [Teradata Aster nPath Visualization Examples](#)

Clickstream Data

Consider a table with clickstream data. The table is defined as:

```
clicks
  ( ts time,
    userid int,
    pageid int,
    category int,
    val float,
    refurl varchar(256)
  )
  DISTRIBUTED BY HASH(category)
```

To consider a sequence of rows for each user, ordered by time, the first clauses of Teradata Aster nPath are written like this:

```
SELECT ...
FROM NPATH(
  ON clicks
  PARTITION BY userid
  ORDER BY ts
  ...
)
...
...
```

For this example, let's define the following symbols:

Table 4 - 99: Symbols for Teradata Aster nPath Example

Symbol	Predicate
A	pageid IN (10, 25)
B	category = 10 OR (category = 20 AND pageid <> 33)
C	category IN (SELECT catid FROM categories GROUP BY catid HAVING COUNT(*) > 10)
D	refurl LIKE '%google%
X	true

To match a symbol shown in the first column, a row must satisfy the associated predicate shown in the second column. In the Teradata Aster nPath syntax, this is written as the SYMBOLS clause:

```
SELECT ...
  FROM nPath
  (
    ...
      SYMBOLS
      (
        pageid IN (10, 25) AS A,
        category = 10 OR (category = 20 AND pageid <> 33 ) AS B,
        category IN
        (
          SELECT catid
            FROM categories
            GROUP BY catid
            HAVING COUNT(*) > 10
        ) AS C,
        refurl LIKE '%google%' AS D,
        true AS X
      )
    ...
  )
  ...
  ...
```

These symbols can now be used with the operators to construct a pattern. The pattern

$A . (B \mid C) + . D? . X^* . A$

for instance, will match a pattern of rows in which the first row satisfies the predicate for A, followed by a non-empty sequence of rows, each satisfying the predicate for B or C, followed by at most one row satisfying the predicate for D, followed by a sequence of arbitrary rows ending at a row satisfying the predicate for A.

Lead

Teradata Aster nPath is also handy for cases when you don't want to match a particular pattern but instead want to create output that combines values from one row in a sequence with values from the next row in the sequence. For example, imagine that you're analyze pageviews on your website, and you want to find out what pageview follows each other pageview. In this example, for each row, we get its pageid as well as the pageid of the next row in sequence:

```
SELECT sessionid, pageid, next_pageid
  FROM nPath(
    ON clicks
    PARTITION BY sessionid
    ORDER BY ts
    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 )
  );
```

Rank

For each row, count the number of preceding rows including this row in a given sequence.

```
SELECT sessionid, pageid, rank
  FROM nPath(
    ON clicks
    PARTITION BY sessionid
    ORDER BY ts DESC
    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 rank )
  )
```

Note the use of DESC in the ORDER BY clause. The reason is that the pattern needs to be matched over the rows preceding the start row, while the semantics dictates that the pattern be matched over the rows following the start row. Reversing the ordering of the rows resolves the issue.

Complex Path Query

Find user click-paths starting at pageid 50 and passing exclusively through either pageid 80 or pages in category 9 or category 10. Find the pageid of the last page in the path and count the number of times page 80 was visited. Report the maximum count for each last page, and sort the output by the latter. Restrict to paths containing at least 5 pages. Ignore pages in the sequence with category < 0.

```
SELECT last_pageid, MAX( count_page80 )
  FROM nPath(
    ON ( SELECT * FROM clicks WHERE category >= 0 )
    PARTITION BY sessionid
    ORDER BY ts
    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 )
  )
  WHERE count_any >= 5
  GROUP BY last_pageid
  ORDER BY MAX( count_page80 )
```

Range-Matching Examples

The examples in this section, which illustrate the use of symbols and ranges in the PATTERN clause, use this table as input:

Table 4 - 100: Input Table (aggregate_clicks)

sessionid	clicktime	userid	productname	pagetype	referrer	productprice
1	07:00:10	333		home	www.yahoo.com	
1	07:00:12	333	iPod	checkout	www.yahoo.com	200.2
1	07:01:00	333	bose	checkout		340
13	15:35:08	67403		home	www.google.com	
35	20:00:01	80000		home	www.godaddy.com	
35	20:02:00	80000	bose	checkout		340
35	20:02:50	80000	itrip	checkout		150
35	20:03:00	80000	iphone	checkout		650
35	20:03:40	80000	ipad	checkout		750
35	20:04:40	80000	oakley	checkout		3400
35	20:05:40	80000	microsoftKB	checkout		170
35	20:06:40	80000	dell	checkout		250
35	20:07:40	80000	dell	home1		250
2	15:34:25	333		home	www.google.com	
14	13:18:30	67403		home	www.google.com	
14	13:18:31	67403		page1		
14	13:18:32	67403		page2		
14	13:18:40	67403	iphone	checkout		650
14	13:19:00	67403	bose	checkout		340
250	20:00:01	80000		home	www.godaddy.com	
250	20:02:00	80000	bose	checkout		340
250	20:02:50	80000	itrip	checkout		150
250	20:03:00	80000	iphone	checkout		650
250	20:03:40	80000	ipad	checkout		750
250	20:04:40	80000	oakley	checkout		340
250	20:05:40	80000	oakley	home1		340

Example 1:

Find the first product that has more than one referrer:

```
SELECT *
FROM npath
(
    ON aggregate_clicks
    PARTITION BY sessionid
    ORDER BY clicktime
    MODE( nonoverlapping )
    PATTERN( 'REFERRER{2,}.NON_REFERRER' )
    SYMBOLS(referrer is not null as REFERRER,
             referrer is null as NON_REFERRER)
    RESULT( first(sessionid of REFERRER) as sessionid,
            first(productname of NON_REFERRER) as product)
) ORDER BY sessionid;
```

This query generates this table as output:

Table 4 - 101: Output Table Example 1

sessionid	product
1	bose

Example 2:

Find all the sessions where the number of products checked out during the session is greater than or equal to 3 and less than or equal to 6 and list the name of the most expensive product, the maximum price for the most expensive product, the name of the least expensive product and the minimum price for the least expensive product:

```
SELECT *
FROM npath
(
    ON aggregate_clicks
    PARTITION BY sessionid
    ORDER BY clicktime
    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)
) ORDER BY sessionid;
```

This query generates this table as output:

Table 4 - 102: Output Table Example 2

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
250	ipad	750	itrip	150

Example 3:

The SQL-MR syntax for this example is the same as the syntax of Example 2, except for the PATTERN clause.

If you wanted to find all the sessions where the number of products checked out during the session is at least 3 the PATTERN would change to:

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

This query generates this table as output:

Table 4 - 103: Output Table Example 3

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
35	oakley	3400	itrip	150
250	ipad	750	itrip	150

In order to find all the sessions where the number of products checked out during the session was exactly 7, the pattern would change to 'H+.D*.C{7}.D'.

Example 4:

The SQL-MR syntax for this example is the same as the syntax of Example 2, except for the PATTERN clause.

An example of a complex query with nested ranges:

```
...
PATTERN( 'H+.D*. (C{2,7}.D){1,4}' )
...
```

This query generates this table as output:

Table 4 - 104: Output Table Example 5

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
35	oakley	3400	itrip	150
250	ipad	750	itrip	150

Multiple Inputs with Partitioned Inputs

Suppose we have an e-commerce store, where we want to find out information about shopper behavior. Specifically, we are interested in substitution of items in the shopping cart preceding a purchase. We want to know if people do more substitution of items (addToCart, followed by removeFromCart) for items that costs more than \$1000 vs. items that costs less than \$100.

The input tables are:

- Purchases table

```
CREATE table purchases_table(
    userid varchar,
    purchaseDate timestamp,
    purchaseId int,
    itemid int,
    numitem int,
    pricePerItem double,
    sessionid int,
    partition key(userid));
```

- AddToCart table

```
CREATE table addToCart_table(
    userid varchar,
    addtocartDate timestamp,
    cartId int, itemid int,
    numitem int,
    pricePerItem double,
    sessionid int,
    partition key(userid));
```

- RemoveFromCart table

```
CREATE table removeFromCart_table(
    userid varchar,
    removefromcartDate timestamp,
    cartId int,
    itemid int,
    numitem int,
    pricePerItem double,
    sessionid int,
    partition key(userid));
```

- ItemViews table

```
CREATE table ItemViews_table(
    userid varchar,
    viewDate timestamp,
    pricedisplayPerItem double,
    sessionid int,
    partition key(userid));
```

We will use the following Teradata Aster nPath function call:

```
SELECT * from npath (
    ON purchases_table PARTITION BY userid ORDER BY purchaseDate
    ON AddToCart_table PARTITION BY userid ORDER BY addToCartDate
    ON RemoveFromCart_table PARTITION BY userid ORDER BY
removeFromCartDate
    ON ItemViews_table PARTITION BY userid ORDER BY viewDate
    MODE('nonoverlapping')

    SYMBOLS(true as PURCHASE,
        AddToCart_table.pricePerItem >= 1000 as expensiveAdd,
        AddToCart_table.pricePerItem <= 100 as cheapAdd,
        RemoveFromCart_table.pricePerItem >= 1000 as expensiveRemove,
        RemoveFromCart_table.pricePerItem <= 100 cheapRemove, true as View)
```

```
PATTERN(' (View*) . (expensiveAdd. (View*) . expensiveRemove) | (cheapAdd. (View*)
) . cheapRemove) Purchase+')

RESULT(
    FIRST(AddToCart_table.itemId of any (expensiveAdd, cheapAdd)) ,
    FIRST(RemoveFromCart_table.itemId of any (expensiveRemove,
cheapRemove)) ,
    FIRST((case when AddToCart_table.pricePerItem >= 1000 then
'expensive' else 'cheap' end) of any (expensiveAdd, cheapAdd))
)
);
```

Multiple Inputs with Dimensional Input

In this example, we want to count the number of different types of advertising impressions leading up to a user clicking on an advertisement. We will count the number of online advertisements viewed by that user and the number of television spots the user *may* have viewed.

Input Tables

impressions Table

A large fact table of online advertising impressions.

```
create table aaf.impressions (userid int, ts date, imp text)
    distribute by hash(userid);
insert into impressions values (1, '2012-01-01', 'ad1');
insert into impressions values (1, '2012-02-01', 'ad1');
insert into impressions values (1, '2012-03-01', 'ad1');
```

clicks Table

A large fact table of user clicks on advertisements.

```
create table aaf.clicks2
(userid int, ts date, click text) distribute by hash(userid);
insert into clicks2 values (1, '2012-04-01', 'ad1');
```

tv_spots Table

A small dimensional table of television advertising spots.

```
create table aaf.tv_spots (ts date, tv_imp text) distribute by
replication;
insert into tv_spots values ('2012-01-01', 'ad1');
```

Syntax

We will look for the pattern `((imp|tv_imp)*.click)` and output the count of impressions and the count of television impressions.

Note that impressions and clicks both have `user_id` as one of their columns. However, `tv_spots` is just a record of television advertisements shown, that any user may have seen. The `tv_spots` table has a timestamp, but no way to identify the `user_id`. In this case, we need to access the `tv_spots` data as a dimensional table, since any user could have viewed any television spot that preceded the click on the online advertisement.

```
SELECT * FROM npath
```

```
(  
    ON impressions PARTITION BY userid ORDER BY ts  
    ON clicks PARTITION BY userid ORDER BY ts  
    ON tv_spots DIMENSION ORDER BY ts  
    MODE('nonoverlapping')  
  
    SYMBOLS(true as imp, true as click,  
            true as tv_imp)  
  
    PATTERN('(imp|tv_imp)*.click')  
    RESULT(COUNT(*) OF imp) AS cnt, COUNT(*) OF tv_imp) AS test  
) ;
```

Output

Table 4 - 105: Input Table

cnt	test
3	1

This example could be made more accurate by increasing the complexity. For example, the tv_spots table could also include zip code information about where the television advertisement was shown. Then we could use Teradata Aster nPath to also check if the zip code matches for each user_id. Then we would only include the television ad impression in the final count if it appeared in the same zip code as the user.

Teradata Aster nPath Visualization Examples

You can use the NpathViz function to visualize nPath output. For nPath visualization examples, see [NpathViz](#).

CHAPTER 5 Statistical Analysis

- Approximate Distinct Count (count_approx_distinct)
- Approximate Percentile (approx_percentile)
- Correlation (stats correlation)
- Histogram
- Enhanced Histogram Function
- Linear Regression (stats linear reg)
- Logistic Regression (deprecated)
- Logistic Predict (deprecated)
- Generalized Linear Model (stats glm)
- Generalized Linear Model Prediction (glmpredict)
- Principal Component Analysis (PCA)
- Simple Moving Average (stats smavg)
- Weighted Moving Average (stats wmavg)
- Exponential Moving Average (stats emavg)
- Volume-Weighted Average Price (stats vwap)
- K-Nearest Neighbor Classification Algorithm (knn)
- Support Vector Machines
- ConfusionMatrix
- Percentile
- Distribution Matching
- LARS Functions
- Sample
- FMeasure

Approximate Distinct Count (count_approx_distinct)

Summary

Based on probabilistic counting algorithms, this function quickly estimates the number of distinct values in a column or combination of columns, while scanning the table only once.

For a column or column combination with large cardinality, it can calculate an approximate count of the distinct values in much less time than would be required to calculate a precise distinct count using SQL's DISTINCT.



Background

For more information about Probabilistic Counting Algorithms, see *Probabilistic Counting Algorithms for Data Base Applications*, by Philippe Flajolet and G. Nigel Martin (<http://portal.acm.org/citation.cfm?id=5215>).

Usage

Permissions

Before running the APPROXD COUNTREDUCE and APPROXD COUNTMAP functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.0)

You call this function by combining a local row function followed by a partition function:

```

SELECT *
  FROM APPROXD COUNTREDUCE
  (
    ON
      (SELECT *
        FROM APPROXD COUNTMAP
        (
          ON {table_name | view_name | (query)}
          COLUMNS ('column_name')
          [ERROR ('error_tolerance')])
        )
      )
    PARTITION BY expression [, ...]
  );
  
```



You may choose to omit the SELECT statements inside internal function calls in SQL-MR, in which case this function has this synopsis:

```
SELECT *
  FROM APPROXDCOUNTREDUCE
  (
    ON
      ( APPROXD COUNTMAP
        (
          ON { table_name | view_name | (query) }
          COLUMNS ('column_name')
          [ERROR ('error_tolerance')]
        )
      )
    PARTITION BY expression [, ...]
  ) ;
```

Arguments

COLUMNS	Required	Specifies the name(s) of the column or columns for which an approximate distinct count will be calculated. This can be any column(s) or combinations of columns, such as, for example: ('col1', 'col2', '(col5:col9)').
ERROR	Optional	Specifies the acceptable error rate, expressed using decimal representation. Ten percent is written as 10, one percent as 1. The default error tolerance rate is ten percent, or 10. Permissible values are any value x, where x is between five one thousandths of a percent and ten percent (that is, $5.0E-4 < x \leq 10$).

Output

The output table has these columns:

Table 5 - 106: Output Table Format

Column Name	Description
column_name	The name of the input column or columns for which the approximate distinct count was computed. Multiple column names are joined with underscores.
cnt	The approximate distinct count.
method	The approach used for calculating the approximate distinct count.

Example

Example Input Data

The example table *page_tracking* contains the columns member_id, page_key, referrer, and pg_seq.

Table 5 - 107: Example input table: page_tracking

member_id	page_key	referrer	pg_seq
1	Home	http://google...	1
1	Profile		2
2	Jobs		1
4	News	http://yahoo...	1
5	Profile		1

Example SQL-MapReduce call

```
SELECT *
  FROM APPROXD COUNTREDUCE
  (
    ON APPROXD COUNTMAP
    (
      ON page_tracking
      COLUMNS ('member_id', 'page_key', '(member_id:page_key)')
      ERROR (1)
    )
    PARTITION BY column_name
  );
```

Example Output

Table 5 - 108: Example output table

column_name	cnt	method
member_id	4	nearExact
page_key	4	nearExact
member_id_page_key	5	nearExact

Error Messages

You may encounter these errors when attempting to run this function:

- ERROR: Maximum error threshold is 10. Select value for error <= 10 or omit error clause.
REASON: Specified error value is greater than the maximum error threshold.
- ERROR: SQL-MR function APPROX_DCOLUMN_PARTIAL failed: Error must be >5.0E-4.
Select value for error >5.0E-4 or omit error clause.
REASON: Specified error values is less than the minimum error threshold.

Approximate Percentile (approx percentile)

Summary

This function computes approximate percentiles for one or more columns of data. The accuracy of the approximation is a parameter the user can vary. Higher accuracy requires longer compute time and vice versa. Optionally, you can specify a column to group by, to compute approximate percentiles over different groups.



Background

The function is based on an algorithm developed by Greenwald and Khanna. It gives e -approximate quantile summaries of a set of N elements, where e is the value you specify as the function's ERROR parameter. Given any rank r , an e -approximate summary returns a value whose rank r' is guaranteed to be within the interval $[r - eN, r + eN]$. The algorithm has a worst case space requirement of $O((1/e) * \log(eN))$.

Usage

This section describes the syntax for using the function, parameter options and data types, and a description of the expected output.

Permissions

Before running the approxPercentileReduce and approxPercentileMap functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (approxPercentileMap version 1.1; approxPercentileReduce version 1.0)

The synopsis below shows the syntax for invoking the approximate percentile function for a column of data, grouping the data by a different set of columns:

```

SELECT *
  FROM approxPercentileReduce
  (
  ON (
    SELECT *
      FROM approxPercentileMap
      (
      ON {table_name | view_name | (query ) }
      TARGET_COLUMN('column_name')
      ERROR( tolerance_value )
      [GROUP_COLUMNS('column_name' [ , ... ] ) ]
      )
    )
  PARTITION BY expression [, ...]
  PERCENTILE(percentile [, ... ] )

```

```
[GROUP_COLUMNS ('column_name' [ , ... ] ) ] );
```

Arguments

<i>TARGET_COLUMN</i>	Required	Specifies the column for which we want to compute the quantile summary. The column must contain data of type smallint, integer, bigint, numeric, real or double precision. This is the column for which you calculate the percentiles.
<i>ERROR</i>	Optional	Specifies the desired accuracy of the approximation. Lower error is more accurate. Must be between .01 and 50. Error of 10 means the quantile will be correct within 10% in either direction. Default value is 1.
<i>PARTITION BY</i>	Required	Specifies the columns by which to partition the output. If the <i>GROUP_COLUMNS</i> argument is used, the column names you specify in this clause must match the names you specify in the <i>GROUP_COLUMNS</i> argument. If the <i>GROUP_COLUMNS</i> argument is not used, you should partition by 1 in the reduce function.
<i>PERCENTILE</i>	Optional	A comma separated list of integers that specifies which approximate percentiles you wish to compute. Default is to compute the quartiles: 0, 25, 50, 75 and 100.
<i>GROUP_COLUMNS</i>	Optional	<p>Specifies the columns to group the data by. Omitting the <i>group_columns</i> clause results in no grouping, and quantiles are computed for the entire column. Note that if you include this clause, you must include it in <i>both</i> the <i>approxPercentileReduce</i> and the <i>approxPercentileMap</i> functions.</p> <p>Columns can be of type varchar or integer. These are group identifying columns. Suppose we have a table with the columns State (varchar), Town (varchar), and Population (integer), and we specify <i>GROUP_COLUMNS</i> ('STATE'). Instead of computing quantiles for Population across all towns, each state would have its quantiles computed individually.</p>

Output

<i>GROUP_COLUMNS</i>	(if <i>group_columns</i> clause was specified) Column(s) specifying which group the percentile belongs to.
<i>PERCENTILE</i>	The percentile we are estimating. For example, percentile: 50 is the median, 75 is the upper quartile, 100 is the maximum.
<i>VALUE</i>	<p>The approximate value of the corresponding percentile, accurate to the degree specified in the <i>ERROR</i> argument.</p> <p>Note that the Approximate Percentile SQL-MR function returns an “approximate” percentile and not the <i>exact</i> percentile. This function will return a more accurate approximate percentile when the dataset is large.</p>

Example

Example Input Data

A sample table called “some_values” with two columns, “segment” and “value”:

Table 5 - 109: Example input table: some_values

segment	value
A	0
A	2
A	4
A	6
A	8
B	1
B	3
B	5
B	7
B	9

Example SQL-MapReduce call:

```
SELECT *
  FROM approxPercentileReduce
  (
    ON
    (
      SELECT *
        FROM approxPercentileMap
        (
          ON some_values
          TARGET_COLUMN( 'value' )
          GROUP_COLUMNS( 'segment' )
          ERROR( 1 )
        )
    )
    PARTITION BY segment
    GROUP_COLUMNS('segment')
    PERCENTILE( 50 )
  ) order by segment;
```

Example Output from Approximate Percentile

Note that the Approximate Percentile SQL-MR function returns an “approximate” percentile and not the *exact* percentile. This function will return a more accurate approximate percentile when the dataset is large, which is not the case in this example.

Table 5 - 110: Example output table

segment	percentile	value
A	50	2
B	50	3

Correlation (stats correlation)

Summary

The correlation functions, CORR_REDUCE and CORR_MAP, compute a global correlation between any pair of columns (COLUMNPAIRS) from a table. You may run this pair of functions on multiple pairs of columns in a single invocation. Measuring correlation allows you to determine if the value of one variable is useful in predicting the value of another.



Usage

Permissions

Before running the corr_reduce and corr_map functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.2)

```

SELECT *
  FROM CORR_REDUCE
  (
    ON CORR_MAP
    (
      ON {table_name | view_name | (query)}
      COLUMNPAIRS ('col1:col2','col2:col3',...)
      KEY_NAME ('key_name')
    )
    PARTITION BY key_name
  );
  
```

Arguments

COLUMNPAIRS	Required	The list of pairs of columns for which correlation will be calculated. Columns whose correlations you calculate must be of type <i>int</i> , <i>bigint</i> , or <i>real</i> . If the columns are of any other compatible type, you must type cast it to one of these three types. To retrieve multiple correlations in a single function invocation, list multiple pairs separated by commas. Each pair is specified as a colon-separated pair, in the form 'col1:col2'. For example, COLUMNPAIRS('col1:col2', 'col2:col3', 'col3:col4').
KEY_NAME	Required	The column name you wish to give to an intermediate column generated by the CORR_MAP function. This intermediate data should then be partitioned on this newly named column before passing it to the CORR_REDUCE function.

Example

Example Input Data

The input table, *income_statistics*, contains the columns:

- participant [int]
- income [bigint]
- years_of_education [real]
- years_of_experience [real]

Table 5 - 111: Example input table: income_statistics

participant	income	years_of_education	years_of_experience
2	100000	20	5
4	35000	16	1
5	41000	18	1
6	29000	12	1
8	24000	12	0
9	50000	16	3
10	60000	17	2
1	125000	19	8
3	40000	16	2
7	35000	14	1

Example SQL-MapReduce call

```
SELECT * FROM CORR_REDUCE(
    ON CORR_MAP(
        ON income_statistics
        COLUMNPAIRS('income:years_of_education',
                    'income:years_of_experience')
        KEY_NAME ('key'))
    PARTITION BY key
) ;
```

Example Output from Correlation Reduce

The example output shows the correlation between the requested columns:

Table 5 - 112: Example output table

Corr	value
Income:Years_of_Education	1
Income:Years_of_Experience	1

Error Messages

You may encounter the following error message when you run this function:

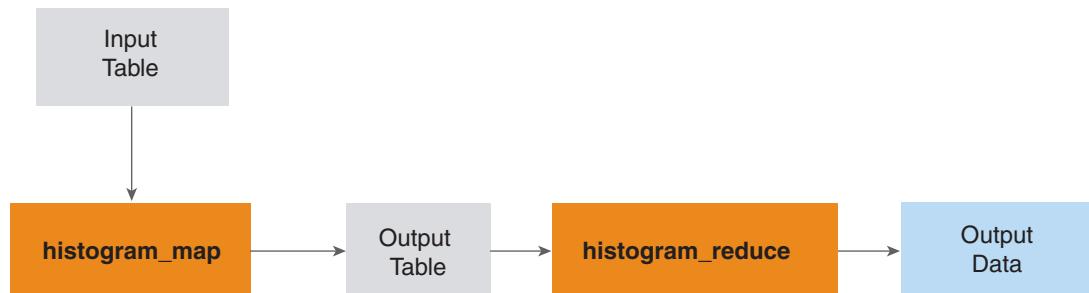
ERROR: COLUMNPairs should be of the form ('X1:Y1','X2:Y2','X3:Y3')

REASON: Columns argument is not specified in the proper format. Each pair should be in the form ('col1:col2','col2:col3','col3:col4') and multiple pairs should be separated by ";" (comma).

Histogram

Summary

The histogram function maps each input row to one or more bins based on criteria you specify and returns the row count for each bin. The SQL-MapReduce histogram function is a combination of SQL-MapReduce row function (*histogram_map*) and an SQL-MapReduce partition function (*histogram_reduce*). The output of the histogram function is useful for assessing the shape of a data distribution.



Usage

In order to generate a histogram on an input data set you must run a map and reduce step on the data. Below is the syntax for running the map phase.

Permissions

Before running the *histogram_reduce* and *histogram_map* functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax of the Map Function (version 1.1)

```
SELECT *
  FROM histogram_map
  (
    ON {table_name | view_name | query}
    VALUE_COLUMN (column_name)
    [ BIN_SIZE ( size of the equi-sized bins) ]
    [ START_VALUE( start value of the first bin) ]
    [ INTERVALS( [<min>:<max>], ... ) ]
    [ BIN_COLUMN_NAME( output-column-name ) ]
    [ START_COLUMN_NAME( output-column-name ) ]
    [ END_COLUMN_NAME( output-column-name ) ]
    [ FREQUENCY_COLUMN_NAME ( output-column-name ) ]
  );

```

Arguments to the Map Function

<i>VALUE_COLUMN</i>	Required	Name of the column the histogram will run on; only one column is permitted.
<i>BIN_SIZE</i>	Optional	For equally sized bins, specifies the width of the bin. Omit this argument if you are <i>not</i> using equally sized bins.
<i>START_VALUE</i>	Optional	For equally sized bins, specifies the minimum (starting) value for the first bin. Omit this argument if you are not using equally sized bins.
<i>INTERVALS</i>	Optional	If the bins are not equi-distant, INTERVALS can be used to specify the minimum and maximum '[<min>:<max>]' of each interval. Note that the bins can be overlapping. This is an alternative to using <i>BIN_SIZE</i> and <i>START_VALUE</i> argument clauses.
<i>BIN_COLUMN_NAME</i>	Optional	Name of output column that shows which bin one or more subject rows were sorted into. Each bin is identified by its bin number. The default name is "bin".
<i>START_COLUMN_NAME</i>	Optional	Name of output column that shows the start (min.) value of this bin. Default name is "start_bin".
<i>END_COLUMN_NAME</i>	Optional	Name of output column that shows the end (max.) value of this bin. Default name is "end_bin".
<i>FREQUENCY_COLUMN_NAME</i>	Optional	Name of output column that shows the count ("local frequency") of rows that sorted into this bin on this partition. Default name is "frequency".

Syntax of the Reduce Function (version 1.1)

```
SELECT *
  FROM histogram_reduce
  (
    ON {table_name|view_name|(subquery)}
    PARTITION BY partition_column
    [ ACCUMULATE ( column_name [, ...] ) ]
    [ FREQUENCY_COLUMN_NAME ( column_name ) ]
  );

```

Arguments to the Reduce Function

<i>ON</i>	Required	Usually the ON clause supplies the results of a <i>histogram_map</i> function.
<i>ACCUMULATE</i>	Optional	List of input columns that will be piped directly through to the output. By default, no input columns are included in the output.

FREQUENCY_COLUMN_NAME Optional Name of the input column that contains the local, per-vworker row counts. This must be the same name you used as the *FREQUENCY_COLUMN_NAME* in *histogram_map*. The default name is “frequency”. (The default values in both functions are same, so you can safely omit this clause.) This name will also be used as the name of the frequency output column.

Output

The histogram function returns one row per bin. Each row describes the bin, shows the count (“frequency”) of rows in that bin, and includes any piped-through columns you specified in *ACCUMULATE* clause.

Example

Example Input Data

Table 5 - 113: Example input table (am_histogram_data)

id	name	age	graduate
100	Henry Cavendish	12	f
200	Sir William	15	f
300	Johann August	19	f
400	Martin Heinrich	20	f
500	Ralph Arthur	25	t
600	Marguerite Catherine	35	t
700	Philip Hauge	40	t
800	Joseph Louis	28	f
900	Marie Curie	12	t

Example Query 1: Fixed-size bins

The first example shows how to define bins of a fixed size using the *BIN_SIZE* and *START_VALUE* clauses:

```
SELECT *
  FROM histogram_reduce(
    ON histogram_map
    (
      ON am_histogram_data
      BIN_SIZE('10')
      START_VALUE('0')
      VALUE_COLUMN('age')
    )
    PARTITION BY bin
    ACCUMULATE('bin','start_bin','end_bin')
  )
  ORDER BY bin;
```

Output of Example Query 1

Table 5 - 114: Example output table

bin	start_bin	end_bin	frequency
1	10	20	4
2	20	30	3
3	30	40	1
4	40	50	1

Example Query 2: Custom-sized bins

The second example shows how to use the *INTERVALS* clause to define each bin individually:

```
SELECT *
  FROM histogram_reduce
  (
    ON histogram_map
    (
      ON am_histogram_data
      VALUE_COLUMN('age')
      INTERVALS('0:30', '20:30', '40:70', '70:100000')
    )
    PARTITION BY bin
    ACCUMULATE('bin', 'start_bin', 'end_bin')
  ) ORDER BY bin;
```

Output of Example Query 2

Table 5 - 115: Example output table

bin	start_bin	end_bin	frequency
0	0	30	7
1	20	30	3
2	40	70	1

Algorithm

The function *histogram_map* is a row function that sorts and counts rows on a per-vworker basis. It reads each row, determines which bin(s) the row belongs to, and updates a locally maintained hashmap. The hashmap's key is the bin number, and its value is the count. If a row belongs to more than one bin (because of overlapping bins) then the function updates the hashmap for all the matched bins. Once all vworker-local rows are processed, the hashmap is complete. The function then emits the counts for all bins in the hashmap.

The function *histogram_reduce* is a partition function that sums each bin's count across all vworkers and emits each bin's global count.

Error Messages

You may encounter the following error messages from the function:

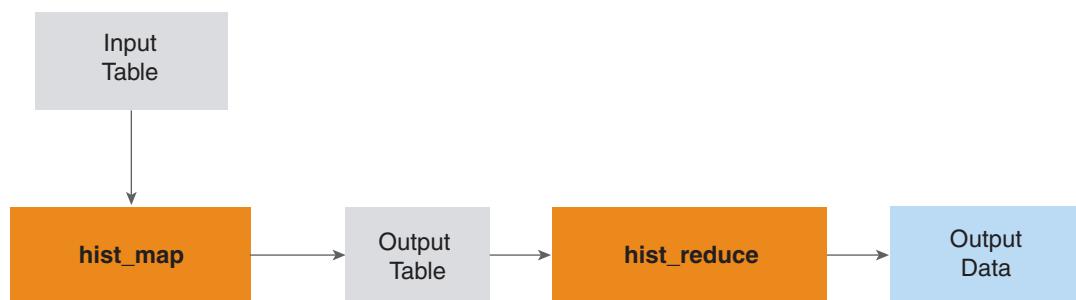
- ERROR: Please specify either the INTERVAL clause or all of the following clauses: BIN_SIZE, START_VALUE.
- ERROR: INTERVALS should be in the form of [<min>:<max>]. <min> should be less than <max>.
- ERROR: INTERVALS should be in the form of [<min>:<max>]. <min> should be less than <max>. For example [10:20].
- ERROR: INTERVALS must have numeric values. For example, [12.3:20].

Enhanced Histogram Function

Summary

The enhanced histogram function (represented by hist_map and hist_reduce) adds these new capabilities over the existing histogram function (represented by histogram_map and histogram_reduce):

- Bin selection—The new function is equipped with algorithms that determine the number of equi-spaced bins without your intervention.
- Non-integer bin ranges—The new function supports non-integer bin ranges.
- Bin IDs and bin ranges in the output columns—By default, the output columns include bin IDs, bin ranges, and bin percentages.
- Bin ranges—The new function supports left-inclusive and right-inclusive bin ranges in the new function.
- Bin breaks—The bin breaks generated by the function are always aligned to multiples of a power of ten or multiples of five times a power of ten.



Implementation Notes

The enhanced histogram function uses the Sturges and Scott Algorithms to compute the number of bins.

hist_map

Usage

Depending on how you use hist_map, there are three syntax options.

Syntax with bin breaks determined by hist_map (version 1.0)

In this option, you let the function decide the bin breaks. For this option to work, you must provide an auxiliary input containing required statistics about the input value streams.

This syntax option uses a SQL query to provide the auxiliary input.

```
SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON (select count(*) as stat_count,
           sum(value_col) as stat_sum,
           sum(value_col*value_col) as stat_squared_sum,
           min(value_col) as stat_min,
           max(value_col) as stat_max
      from data_table where value_col is not null and
           not value_col = 'infinity' and not value_col = '-infinity')
) as data_stat DIMENSION
BIN_SELECT('Sturges'|'Scott'|number_of_bins)
VALUE_COLUMN('value_col')
[INCLUSION('left'|'right'))];
```

This syntax option uses hist_prep, a data-preparation SQL-MR function to provide the auxiliary input.

```
SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON hist_prep(
        ON data_table VALUE_COLUMN('value_col') as data_stat DIMENSION
        BIN_SELECT('Sturges'|'Scott'|number_of_bins)
        VALUE_COLUMN('value_col')
        [INCLUSION('left'|'right'))]);
```

You can choose one of two algorithms to use: Sturges or Scott. Or you can specify an approximate number of bins using the argument clause *BIN_SELECT*.

Syntax with bin breaks specified (version 1.0)

In this option, you specify the bin breaks. If the input table bin_breaks exists, the function bypasses the preprocessing phase. The bin breaks should contain the boundary values of the histogram and hence it is required that the bin breaks contain at least two valid numerical values.

```
SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON bin_breaks DIMENSION
    [INCLUSION('left'|'right'))]
    VALUE_COLUMN('value_col')
    BREAKS_COLUMN('breaks_col')
);
```

Syntax with bin start and size specified (version 1.0)

In this option, you specify the bin start and bin size, with an optional bin end. If these arguments exist, the function bypasses the preprocessing phase.

```
SELECT * FROM hist_map(
    ON data_table
    STARTVALUE('bin_start')
    BINSIZE('bin_size')
    ENDVALUE('bin_end')
    [INCLUSION('left' | 'right')]
    VALUE_COLUMN('value_col')
) ;
```

hist_reduce

Usage

Syntax (version 1.0)

```
SELECT * FROM hist_reduce(
    ON hist_map(...) PARTITION BY 1
) ;
```

Example

Example Input

Table 5 - 116: Input table: hist_input_double

id	val
2	-10.089935025935
14	-10.6228491988133
42	-9.59265712307246
54	-10.5319644153708
70	-10.3406983323497
82	-10.1321622550667
122	20.2839263193374
138	15.8374951463775
9	-9.81850699582376
21	-10.0042862032006
33	-9.529420723528
97	-10.1994285890814
109	27.2615641537664

Table 5 - 116: Input table: hist_input_double (continued)

id	val
121	19.1488810622712
129	16.730491256619
25	-9.67368305247125
37	-9.97833166966823
49	-9.72993472687019
61	-10.1544184436428
73	-10.213461706534
145	13.7188860701111
1	-9.93901248866905
13	-10.0763996341356
77	-10.4363519844653
89	-10.0085597938187
101	23.4809806203462
113	16.6133660757841
133	22.2352006129263
6	-9.8641824777226
18	-9.52670242751413
30	-9.83543558729657
66	-10.5151002886659
78	-10.649148777438
150	18.0172754609913
58	-10.1349115458742
94	-10.5023878074091
106	22.2302198909164
118	18.2204977106797
91	-10.6066354124338
119	20.601841433874
135	21.5529249912927
147	22.2073915333846
38	-10.034131753544

Statistical Analysis
Enhanced Histogram Function

Table 5 - 116: Input table: hist_input_double (continued)

id	val
50	-10.7535826788157
62	-9.38394426569233
26	-10.0224247629395
74	-9.80242511068245
90	-10.0446449115793
86	-10.387450271409
102	16.7927691502397
98	-10.0283864186185
114	21.5315753567816
110	20.1109205244893
126	18.908613273394
134	20.4674797456188
146	22.8729872133318
64	-9.03999365846491
100	-9.72605696156186
112	25.2758715828149
124	22.4221344761528
17	-9.82538975563841
29	-10.4080231461093
41	-9.98703227695738
53	-9.37110639478968
65	-10.257157753386
137	15.400668997392
149	17.2185097396521
45	-9.85466118862472
57	-10.1823485753786
69	-10.1882580288656
81	-9.74058738424502
93	-10.5112844524682
105	21.4540178403129

Table 5 - 116: Input table: hist_input_double (continued)

id	val
117	17.3234160313955
8	-9.87536033614274
20	-10.4300732498497
36	-9.48457194669718
48	-9.83635980043442
76	-9.89776731015737
88	-10.3979963186058
116	22.7766462433654
132	12.7751893386632
144	20.0348784917977
5	-9.54601010580263
85	-9.70039049728767
125	19.5646545555243
141	21.0170186774825
4	-10.5157619927868
16	-9.66595162903024
28	-9.59024539703493
40	-10.6787432078584
52	-10.3157662406915
136	16.6706474095839
148	16.886710246976
84	-9.7053808728831
96	-9.74976364452532
108	27.3365377748766
120	17.6063862074198
128	23.6394048514914
140	24.470261539068
19	-9.9632865740983
31	-9.6691898336834
43	-9.9980587038102

Statistical Analysis

Enhanced Histogram Function

Table 5 - 116: Input table: hist_input_double (continued)

id	val
55	-9.45118577985813
67	-9.83938235956604
79	-9.81604328860848
139	23.2942235755788
32	-11.0481456249677
44	-10.4642166340907
56	-10.2141550952978
68	-9.31840349490339
80	-10.5487846288416
92	-10.129878561544
104	19.4212178455048
10	-10.4170751906105
22	-10.2529612620062
34	-9.75559036678964
46	-9.76583979623948
130	19.602187751865
142	15.8646207711702
11	-9.76121153568807
23	-10.3263912740189
35	-9.5121552432308
47	-10.121022008946
59	-10.1848597235944
131	19.3663559081593
143	19.1238168621344
3	-8.99082078646726
15	-9.45131423692489
27	-10.4134969625979
39	-9.77150067783499
103	22.3434835350417
115	22.5534031483621

Table 5 - 116: Input table: hist_input_double (continued)

id	val
127	22.2814226272086
12	-9.92763578556779
24	-9.414409677841
60	-9.69963305644533
72	-10.9358037425005
7	-9.84451596365069
71	-9.76714478089298
83	-10.3040789360278
95	-9.55628326564157
107	22.2649342064299
51	-9.89471049195621
63	-9.99363218686126
75	-9.70333062458375
87	-10.4149712586114
99	-9.61816138248268
111	17.5852905090833
123	15.3136110957096

Example SQL-MR Call

```
select * from hist_reduce(
  on hist_map(
    on hist_input_double as data_input partition by any
    on hist_prep(on hist_input_double value_column('val')) as data_stat dimension
    value_column('val')
    bin_select('biScott'))
  ) partition by 1) order by bin;
```

Example Output

The output contains five columns: bin, bin_start, bin_end, bin_count, and bin_percent. An example output table looks like this:

Table 5 - 117: Example output

bin	bin_start	bin_end	bin_count	bin_percent
0	-20	-10	47	31.33
1	-10	0	53	35.33

Table 5 - 117: Example output (continued)

bin	bin_start	bin_end	bin_count	bin_percent
2	0	10	0	0.00
3	10	20	24	16.00
4	20	30	26	17.33

Linear Regression (stats linear reg)

Summary

Outputs coefficients of the linear regression model represented by the input matrices. The zeroth coefficient corresponds to the slope intercept.



Usage

Permissions

Before running the linreg and linregmatrix functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.0)

```
SELECT *
FROM LINREG
(
    ON LINREGMATRIX
    (
        ON {table_name | view_name | (query)}
    )
    PARTITION BY 1
);
```

Assumptions

All the data should be submitted to one worker which means that user needs to perform a "PARTITION BY 1". It is also assumed that the Y component of the data point is provided in the last input column.

Example

Example Input Data

The sample table *data_set* contains:

- X1 [int]
- X2 [int]
- Y [int]

Table 5 - 118: Example input table: table data_set

X1	X2	Y
300	1000	30000
300	3000	10000
100	1000	10000
100	2000	20000
100	3000	30000
200	1000	20000
200	2000	10000

Example SQL-MapReduce call

```
SELECT *
  FROM LINREG
  (
    ON LINREGMATRIX
    (
      ON data_set
    )
  PARTITION BY 1
) ;
```

Example Output from Linear Regression

Table 5 - 119: Example output table

coefficient_index	value
0	19696.9696969697
1	-3.03030303030308
2	-0.303030303030308

Notes

Please note that all the rows should be provided to one worker. Hence "PARTITION BY 1" should be used.

Error Messages

You may encounter the following errors when using this function:

ERROR: The input data results in a singular matrix and hence there is no solution. The input data set provided is perfectly linear.

If two or more input columns are co-linear, or very closely correlated, then no solution to linear regression exists, so the function will fail. Looking at correlations between columns using Aster Database's [Correlation \(stats correlation\)](#) function can help uncover sources of co-linearity. Removing co-linear columns should resolve the issue.

Logistic Regression (deprecated)

This function has been deprecated, and should not be used. Use the "[Generalized Linear Model \(stats glm\)](#)" on page 205 function instead.

Logistic Predict (deprecated)

This function has been deprecated, and should not be used. Use the "[Generalized Linear Model Prediction \(glmpredict\)](#)" on page 220 function instead.

Generalized Linear Model (stats glm)

Summary

Generalized linear model (GLM) is an extension of the linear regression model that enables the linear equation to be related to the dependent variable(s) via a link function. GLM performs linear regression analysis for any of a number of distribution functions using a user-specified distribution family and link function. The link function is chosen based upon the distribution family used and the assumed nonlinear distribution of expected outcomes. Supported link models in Aster Database are ordinary linear regression, logistic regression (logit model), and Poisson log-linear model.



A GLM has three parts:

- 1 A random component - the probability distribution of Y from the exponential family
- 2 A fixed linear component - the linear expression of the predictor values (X_1, X_2, \dots, X_p), expressed as η or $X\beta$
- 3 A link function that describes the relationship of the distribution function to the expected value of Y (for example, linear regression, logistic or logit regression, or Poisson loglinear model)

GLM adds support to categorical variables. In the following table "outcome" is the dependent variable, and "weight," "color," and "size" are the independent variables, where "weight" is a quantitative variable and "color" is a qualitative one.

In regression analysis, the qualitative variable is called categorical (or dummy) variable. If a variable is a categorical one and has n category values, normally it need to be extended into n-1 dummy variables where each one is a binary indicator for one of the category value.

For example, the “color” variable will be extended into two variables “color_yellow” (1 if color=“yellow” and 0 otherwise) and “color_blue” (1 if color=“blue” and 0 otherwise). If color is red, then both “color_yellow” and “color_blue” will be 0.

Table 5 - 120: Categorical variables

weight	color	outcome
10	yellow	1
5	blue	0
6	red	1

Background

The table below, from Venables and Ripley 2002, pages 184-185, describes the common Families and Link Functions. In the table, 'D' denotes the default link for each family.

Table 5 - 121: Common Link Functions

Link	Symbol in GLM	binomial (logistic)	Gamma	Gaussian	inverse-Gaussian	Poisson	negative-binomial
logit	LOGIT	D				*	*
probit	PROBIT	*					
cloglog	COMPLEMENTARY_LOG_LOG	*					
identity	IDENTITY		*	D	*	*	*
inverse	INVERSE		D	*	*		
log	LOG	*	*	*	*	D	D
$\frac{1}{\mu^2}$	INVERSE_MU_SQUARED				D		
sqrt	SQUARE_ROOT					*	

Table 5 - 122: Common Families Functions

Family	Symbol in GLM	Canonical link	Name	Variance function
binomial (logistic)	BINOMIAL/LOGISTIC	$\log \frac{\mu}{1-\mu}$	logit	$\mu(1-\mu)$
Poisson	POISSON	$\log \mu$	log	μ

Table 5 - 122: Common Families Functions

Family	Symbol in GLM	Canonical link	Name	Variance function
Gaussian	GAUSSIAN	μ	identity	σ^2

More information on the canonical links follows:

- Binomial (or logistic) regression is used when the dependent variable (Y) has only two different possible values (0 and 1, "yes" and "no", "true" and "false"). The analysis applies the model to the data and predicts the most likely of the two possible outcomes for each input. A logit, or logarithm of odds is supplied for each outcome.
- Poisson regression is used to model count data (non-negative integers) and contingency models (matrices of the frequency distribution of variables). It assumes that the dependent variable (Y) has a Poisson distribution. A Poisson distribution segments the data into intervals (For example, of time, geographic location, etc.) It then calculates the discrete probability of one or more events occurring within these segments. The logarithm link is used.
- Gaussian regression happens when the data is grouped around a single mean, and can be graphed in a "normal" or bell curve distribution.

Usage

Permissions

Before running the `glm`, `glm_reducesolveandupdate`, `glm_reducebymatrixindex`, `glm_map`, and `glm_reduceasympoticstats` functions, you must obtain the right to run them using the `GRANT EXECUTE` command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.2)

```
SELECT *
FROM GLM (
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    [COLUMNNAMES('column_names')]
    [CATEGORICALCOLUMNS('columnname_value_pair'[,...])]
    [FAMILY('family')]
    [LINK('link')]
    [WEIGHT('weight_column_name')]
    [THRESHOLD('threshold')]
    [MAXITERNUM('max_iterations')]
) ;
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [::1]:2406. The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	Input table is a table with several columns containing the list of features by which we are clustering the data. One of them is the response, and some other columns are the predictors. There can also be columns used as 'weight' or 'offset'.
<i>OUTPUTTABLE</i>	Required	Specify a name for the output table for the coefficients. This table must not exist, so if it does exist, you must DROP it before running the function again. For Stats GLM, the output is written to the screen, and the output table is the table where the coefficients are stored.
<i>COLUMNNAMES</i>	Optional	<p>The dependent variable column name followed by the predictor column names from the input table. The list of input column names must be in this format: 'Y,X1,X2,...,Xp'.</p> <p>By default, all columns except the one specified in the WEIGHT clause are used as input, where the first column is the dependent variable column, and the remaining columns are the predictor columns.</p>

<i>CATEGORICALCOL</i>	Optional	<p>List of columnname_value pairs. Each pair contains the name of a categorical input column and the corresponding category values, which appear in the model generated by GLM.</p> <p>This argument provides support for categorical variables (or dummy variables). Assuming one column is a categorical input that contains n different values (v_1, v_2, \dots, v_n), then, in the model fitting process, this one column of input is expanded to $n-1$ dummy variables (corresponding to $v_1 \sim v_{n-1}$, where each variable has a value of 1 or 0 to indicate the presence of a corresponding category value. Rows that have the value v_n in the categorical column take the value 0 for all the $n-1$ dummy variables. In that case, the v_n category is called an omitted category. The reason for only keeping $n-1$ variables is to avoid the dummy variable trap.</p> <p>You can specify columnname-value pairs in one of these forms:</p> <ul style="list-style-type: none"> • '<i>columnname:max_cardinality</i>'—Sometimes there may be more than 1000 categorical values in one column and you just want to use a few of the most common categories and put the remaining ones into the “others” category. To do so, you set the maximum cardinality for that column. For example, if you specify '<i>column_a:3</i>', this means that for <i>column_a</i>, keep the 3 most common categories, and for the rows that do not belong to those 3 categories, set their category to “others.” • '<i>columnname:(category_value1[, category_value2])</i>'—In this case, you can specify the list of categories to keep one column. The remaining categories are merged into the “others” category. For example, '<i>column_a : (red, yellow, blue)</i>'. • '<i>columnname</i>'—If you only provide the column name, all category values appear in the model. <p>If the <i>COLUMNNAMES</i> clause is specified, the columns in <i>CATEGORICALCOLUMNS</i> must appear in the <i>COLUMNNAMES</i> clause.</p> <p>The categorical columns specified in this clause (<i>CATEGORICALCOLUMNS</i>) should be of type character, integer, boolean, date, time (without time zone), timestamp (without time zone), or ip4.</p>
<i>FAMILY</i>	Optional	The default value is 'BINOMIAL', which is equivalent to 'LOGISTIC'. The allowed families are: BINOMIAL/LOGISTIC, POISSON, and GAUSSIAN.
<i>LINK</i>	Optional	The default value is 'CANONICAL'. The canonical link functions and the link functions that are allowed for a particular exponential family are listed in the table Table 5 - 121 .
<i>WEIGHT</i>	Optional	<p>The default value is '1'. You may specify an integer, or a column name in the input table whose type must be real or integer. The 'weight' is primarily used to assign some weight to each response. You may find the following explanation in R useful:</p> <p>“Non-‘NULL’ ‘weights’ can be used to indicate that different observations have different dispersions (with the values in ‘weights’ being inversely proportional to the dispersions); or equivalently, when the elements of ‘weights’ are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations. For a binomial GLM prior weights are used to give the number of trials when the response is the proportion of successes: they would rarely be used for a Poisson GLM.”</p>
<i>THRESHOLD</i>	Optional	Specify the convergence threshold. The default value is 0.01.
<i>MAXITERNUM</i>	Optional	This is the maximum number of iterations that the algorithm will run before quitting if the convergence threshold has not been met. The default value is 25.

Input

In the *COLUMNNAMES* argument, you pass the list of columns containing the variables. You must pass the dependent variable first, before passing the predictors. You pass the input column names as a comma-separated list, as shown in this example:

```
COLUMNNAMES ('Y', 'X1', 'X2', 'X3', 'X4', 'X5')
```

The example above describes input columns of the following form:

Y	X1	X2	X3	X4	X5	X6
Y	X1	X2	X3	X4	X5	X6
Y	X1	X2	X3	X4	X5	X6

The first column represents the dependent variable, and rest of the columns represent the predictors.

The *COLUMNNAMES* argument in GLM as an optional argument. If the *COLUMNNAMES* argument is not specified, all columns (except the one specified by the *WEIGHT* argument) will be used.

There may optionally be one or more other columns containing values used as a 'weight' or 'offset'. In the *WEIGHT* argument, you can optionally pass a weight as an integer.

Alternatively, you can specify a column that contains weight values for each record. The column values must be of the type real or integer. The weight column is used to assign a relative weight to each response. An example of the use of weight or offset might be to give data that were observed more than ten years ago a lower weight in the calculation, and gradually increase the weight until the data are current. Another example might be to weight product reviews more heavily by reviewer based on the number of reviews submitted or on the number of followers.

Onscreen Output

The onscreen output of the Stats GLM function is a regression analysis of the data, using the family and link functions specified.

Output Columns

When a particular column is not used for its corresponding row, the column will contain a value of zero (0). The following is a description of the columns that appear in the output:

Table 5 - 123: GLM onscreen output columns

Column	Description
predictor	This column contains the column name for each predictor that was input to the function. It is also used to label the other (non-predictor) rows (Intercept, ITERATIONS#, ROWS#, Residual deviance, AIC and BIC).
estimate	The mean of the supplied values for each predictor.
std_error	Standard deviation of the mean for each predictor (standard error).

Table 5 - 123: GLM onscreen output columns

Column	Description
z_score	The z-score is a measure of the likelihood that the null hypothesis is true, given this sample. It is derived by taking the difference between the observed sample mean and the hypothesized mean, divided by the standard error.
p_value	The significance level (p-value) for each predictor.
significance	The likelihood that the predictor is significant. The most significant predictor in the analysis will be designated with “***”. Less significant predictors will be designated with “.”, and variables without significance will have nothing in this column.

Output Rows

The output includes a row for each of these parameters with a value for estimated value, standard error, z-score, p-value, and significance:

Table 5 - 124: Output row parameters

Parameter	Description
Intercept	The value of the logit (Y) when all predictors are 0.
Predictors	A row for each predictor value (X1,X2,...,Xp).

The following values are also output in the second column (estimate). The description is given for each item below in the last (significance) column:

Table 5 - 125: Output values in the estimate column

Value	Description
ITERATIONS#	The number of Fisher Scoring iterations performed on the function.
ROWS#	The number of rows of data received as input.
Residual deviance	The deviance, with degrees of freedom noted in the significance column.
AIC	Akaike information criterion.
BIC	Bayesian information criterion.

The coefficients are also stored in the table *outputtable* for later use.

Output Table

The output table specified by the OUTPUTTABLE argument stores the estimated coefficients and statistics, which are used by the GLMPREDICT function.

Output Table Columns

When a particular column is not used for its corresponding row, the column contains a value of zero (0). This is a description of the columns that appear in the output table:

Table 5 - 126: GLM output table columns

Column	Description
attribute	The index of each predictor, starting from 0.
predictor	The column name for each predictor that was supplied as input to the function.
category	The category names of each predictor. Numeric predictors have NULL values in this column.
estimate	The mean of the supplied values for each predictor.
std_error	Standard deviation of the mean for each predictor (standard error).
z_score	The z-score is a measure of the likelihood that the null hypothesis is true, given this sample. It is derived by taking the difference between the observed sample mean and the hypothesized mean, divided by the standard error.
p_value	The significance level (p-value) for each predictor.
significance	The likelihood that the predictor is significant. The most significant predictor in the analysis will be designated with “***”. Less significant predictors will be designated with “.”, and variables without significance will have nothing in this column.

Output Table Rows

The output includes a row for each of the following with a value for estimated value, standard error, z-score, p-value, and significance:

Table 5 - 127: Output table parameters

Parameter	Description
Intercept	The value of the logit (Y) when all predictors are 0.

Table 5 - 127: Output table parameters (continued)

Parameter	Description
Predictors	<p>A row for each predictor value (X1,X2,...,Xp). Each numeric input column corresponds to one predictor. For categorical input columns, the numbers of predictors are depends on CATEGORICALCOLUMNS argument:</p> <ul style="list-style-type: none"> If the input is in this format: '<i>columnname:n</i>', then <i>n+1</i> predictors are generated, where <i>n</i> predictors correspond to the <i>n</i> selected dummy variables, and the (<i>n+1</i>)-th predictor has the value “@others” in the category column. If the input is in this format: '<i>columnname:(v₁,v₂,...,v_n)</i>', then <i>n+1</i> predictors are generated, where <i>n</i> predictors correspond to the <i>n</i> selected dummy variables, and the (<i>n+1</i>)-th predictor has the value “@others” in the category column. If the input is in this format: '<i>columnname</i>', then <i>k</i> predictors are generated, where <i>k</i> is the number of distinct values in that column. <p>During model fitting, one predictor is omitted (to avoid the dummy variable trap) for a categorical input column. Because of that, the omitted predictor has a NULL value in these columns: estimate, std_err, z_score, p_value, and significance. For the first two kinds of input, the category “@others” is omitted, and for the third kind of input, the category with the most population is omitted.</p>

Examples

Logistic Regression Analysis Example

Example Input Data

The example table shows the temperature and the level of damage recorded at each temperature for a piece of equipment.

Table 5 - 128: Example input table: glm_test1

temp	damage
53	5
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0

Table 5 - 128: Example input table: glm_test1 (continued)

temp	damage
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	1
76	0
76	0
78	0
79	0
81	0

Example SQL-MapReduce call

Note that for logistic regression, you do not need to specify FAMILY and LINK, even though those are shown in this example. The function will perform logistic regression when these are left as their defaults. You may also leave out the WEIGHT argument.

```
SELECT * FROM GLM (
    ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('glm_test1')
    outputTable('glm_output1')
    columnNames('damage', 'temp')
    family('LOGISTIC')
    link('CANONICAL')
    weight('6')
    threshold('0.01')
    maxIterNum('10')
) ;
```

Example Output

The output of the stats GLM function is a regression analysis of the data, using the family and link function(s) specified. The table is output to the screen, and has the following format:

Table 5 - 129: Example output table

predictor	estimate	std_error	z-score	p_value	significance
(Intercept)	11.663	3.29627	3.53823	0.000402812	***
temp	-0.216234	0.0531772	-4.06628	4.7769e-05	***
ITERATIONS #	6	0	0	0	Number of Fisher Scoring iterations
ROWS #	23	0	0	0	Number of rows
Residual deviance	16.9123	0	0	0	on 21 degrees of freedom
AIC	33.6748	0	0	0	Akaike information criterion
BIC	35.9458	0	0	0	Bayesian information criterion

The coefficients are also stored in the 'glm_output1' table for later use. You can use a SELECT statement to view them:

```
SELECT * FROM glm_output1;
```

Table 5 - 130: Example output table: glm_output1

attribute	predictor	estimate	std_error	z-score	p_value	significance
0	(Intercept)	11.663	3.29627	3.53823	0.000402812	***
1	temp	-0.216234	0.0531772	-4.06628	4.7769e-05	***

Input Table with Categorical Variables Example

Example input data

Table 5 - 131: Input table (glm_test2)

id	col1	col2	col3	col4	response
1	female	57	yes	14	0
2	female	37	yes	18	0
3	female	22	no	14	1
4	male	27	yes	16	0
5	male	42	yes	20	0
6	female	27	no	17	1
7	male	27	yes	17	1
8	female	32	no	20	1
9	male	27	no	18	0
10	male	42	yes	17	0
11	male	37	yes	20	1
12	female	22	no	12	0
13	male	52	yes	20	1
14	female	42	yes	12	1
15	male	57	yes	20	0
16	female	22	no	12	1
17	female	32	no	18	0
18	male	32	yes	12	1
19	male	32	no	18	0
20	female	27	yes	16	0
21	male	42	yes	17	0
22	female	22	no	16	0
23	female	22	no	14	0
24	male	27	no	17	0
25	female	27	no	16	0

Example SQL-MapReduce call

```
SELECT * FROM GLM (
    ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('glm_test2')
    outputTable('glm_output2')
    columnNames('response','col1', 'col2', 'col3', 'col4')
    categoricalColumns('col1', 'col3 : (yes)')
    family('LOGISTIC')
    link('CANONICAL')
    weight('1')
    threshold('0.01')
    maxIterNum('10')
) ;
```

Example output

Table 5 - 132: Onscreen output

predictor	estimate	std_error	z_score	p_value	significance
(Intercept)	1.09303	2.8509	0.383399	0.701424	
col1.male	-0.225616	1.05708	-0.213433	0.83099	
col2	-0.0264402	0.0586266	-0.450994	0.651994	
col3.yes	0.743653	1.24071	0.599379	0.54892	
col4	-0.0658533	0.192161	-0.342699	0.731825	
ITERATIONS #	3	0	0	0	Number of Fisher Scoring iterations
ROWS #	25	0	0	0	Number of rows
Residual deviance	31.9072	0	0	0	on 20 degrees of freedom
AIC	41.9072	0	0	0	Akaike information criterion
BIC	48.0016	0	0	0	Bayesian information criterion

To display the output table (glm_output2):

```
SELECT * FROM glm_output2 order by attribute;
```

Table 5 - 133: Output table (glm_output2)

attribute	predictor	category	estimate	std_err	z_score	p_value	significance
0	(Intercept)		1.09303	2.8509	0.383399	0.701424	

Table 5 - 133: Output table (glm_output2) (continued)

attribute	predictor	category	estimate	std_err	z_score	p_value	significance
1	col1	female					
2	col1	male	-0.225616	1.05708	-0.213433	0.83099	
3	col2		-0.0264402	0.0586266	-0.450994	0.651994	
4	col3	@@others					
5	col3	yes	0.743653	1.24071	0.599379	0.54892	
6	col4		-0.0658533	0.192161	-0.342699	0.731825	

Comparison with R

```

> input<-read.table("glm_test2.csv", header=TRUE, sep=", ")
> input
   id   col1  col2  col3  col4 response
1  1 female   57   yes   14      0
2  2 female   37   yes   18      0
3  3 female   22    no   14      1
4  4 male     27   yes   16      0
5  5 male     42   yes   20      0
6  6 female   27    no   17      1
7  7 male     27   yes   17      1
8  8 female   32    no   20      1
9  9 male     27    no   18      0
10 10 male    42   yes   17      0
11 11 male    37   yes   20      1
12 12 female  22    no   12      0
13 13 male    52   yes   20      1
14 14 female  42   yes   12      1
15 15 male    57   yes   20      0
16 16 female  22    no   12      1
17 17 female  32    no   18      0
18 18 male    32   yes   12      1
19 19 male    32    no   18      0
20 20 female  27   yes   16      0
21 21 male    42   yes   17      0
22 22 female  22    no   16      0
23 23 female  22    no   14      0
24 25 female  27    no   16      0
25 24 male    27    no   17      0
> model<-glm(response~col1+col2+col3+col4,
family=binomial(link="logit"), data=input)
> summary(model)
Call:
glm(formula = response ~ col1 + col2 + col3 + col4, family =
binomial(link = "logit"),
  data = input)
Deviance Residuals:
    Min      1Q  Median      3Q     Max
-1.2069 -0.9383 -0.8032  1.2732  1.6571
Coefficients:
              Estimate Std. Error z value Pr(>|z|)

```

```
(Intercept) 1.09303 2.85090 0.383 0.701
col1male -0.22562 1.05708 -0.213 0.831
col2 -0.02644 0.05863 -0.451 0.652
col3yes 0.74365 1.24071 0.599 0.549
col4 -0.06585 0.19216 -0.343 0.732
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 32.671 on 24 degrees of freedom
Residual deviance: 31.907 on 20 degrees of freedom
AIC: 41.907
Number of Fisher Scoring iterations: 4
```

Error Messages

You may see the following errors:

- ERROR: SQL-MR function GLM_MAP failed unexpectedly. The following is information that may be useful to the developer of GLM_MAP: org.apache.commons.math.MathRuntimeException\$4: the Poisson mean must be positive (0).
REASON: The Poisson mean is negative. This type of link cannot be applied to this data.
- ERROR: SQL-MR function GLM failed: The table glm_output1 already exists.
REASON: You must drop the output table before running the function.
- ERROR: SQL-MR function GLM failed: Connection to jdbc:ncluster://10.50.129.100/beehive could not be established.
REASON: JDBC connection could not be made. Test your JDBC connection.
- ERROR: SQL-MR function GLM failed: No columns in the input table: 'input_tabel_name'.
REASON: The error happens when there is only one column in the input table while which is used as WEIGHT column.
- ERROR: The predictor number (2000) exceeds the limit. Currently the maximum predictor number allowable is 1598.
REASON: The maximum number of supported predictors is 1598, which means that the number of numeric predictor columns plus the category number specified in the CATEGORICALCOLUMNS clause cannot exceed 1598.
- ERROR: The category values 'black', 'white' aren't found in column 'color'
REASON: Some of the values specified in the CATEGORICALCOLUMNS do not exist in the categorical columns.
- ERROR: The categorical column "color" isn't specified in 'columnNames' clause.
REASON: The column names specified in CATEGORICALCOLUMN clause must appear in the COLUMNNAMES clause if the latter one is used.
- ERROR: The predictor 'columnname' should be of type real, numeric, integer, or boolean.
REASON: The data type of a predictor column that is not a categorical one should be one of the above types.
- ERROR: The predictor 'columnname' specified in 'CATEGORICALCOLUMNS' clause should be of type character, integer, boolean, date, time (without time zone), timestamp (without time zone), or ip4.
REASON: The data type of a categorical column should be one of above types.

Generalized Linear Model Prediction (glmpredict)

Summary

This function lets you use the model generated by the Stats GLM function to predict new input data.



Usage

Syntax (version 1.2)

```

SELECT * FROM GLMPREDICT(
    ON input_table
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    [PASSWORD('password')]
    MODELTABLE('model_table_name')
    [ACCUMULATE('column_names')]
    FAMILY('family')
    [LINK('link')]
) ;
  
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host;port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [::1]:2406. The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default is "beehive".
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>MODELTABLE</i>	Required	The second input table for this function is the model table generated by the Stats GLM function.
<i>ACCUMULATE</i>	Optional	A comma-separated list of the names of the columns in the input table that will be displayed in the output. For example: ACCUMULATE ('column1', 'column2', 'column3').
<i>FAMILY</i>	Required	The default value is 'BINOMIAL', which is equivalent to 'LOGISTIC'. The allowed families are: BINOMIAL/LOGISTIC, POISSON, GAUSSIAN, GAMMA, INVERSE_GAUSSIAN, NEGATIVE_BINOMIAL.
<i>LINK</i>	Optional	The default value is 'CANONICAL'. The canonical link functions and the link functions that are allowed for a particular exponential family are listed in the table " Common Link Functions " on page 206.

Output

The onscreen output of the Stats GLM predict function is a table that consists of the columns specified in the ACCUMULATE clause in addition to the fitted_value column, which contains the score of the input data. You can treat the score as the prediction of the mean of certain exponential family.

Example

Example Input Data

The input tables to glmpredict in this example are:

- [Table 5 - 130](#): Shows the model table generated by the Stats GLM function.
- [Table 5 - 134](#): Shows the temperature and the level of damage recorded at each temperature for a piece of equipment.

Table 5 - 134: Example input table: glm_damage_index

id	temp	damage
1	53	5
2	57	1
3	58	1
4	63	1
5	66	0
6	67	0
7	67	0
8	67	0
9	68	0
10	69	0
11	70	1
12	70	0
13	70	1
14	70	0
15	72	0
16	73	0
17	75	0
18	75	1
19	76	0
20	76	0

Table 5 - 134: Example input table: glm_damage_index (continued)

id	temp	damage
21	78	0
22	79	0

Example GLMPREDICT Call

```
SELECT * FROM GLMPREDICT (
    ON glm_damage_index
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    MODELTABLE ('glm_output1')
    ACCUMULATE ('id', 'temp', 'damage')
    FAMILY('LOGISTIC')
    LINK('LOGIT')
)
ORDER BY temp, damage;
```

Example Output of Stats GLMPREDICT

This table displays the output of the Stats GLMPREDICT function. The fitted_value column displays the score of the input data.

Table 5 - 135: Example output table

id	temp	damage	fitted_value
1	53	5	0.550476961762949
2	57	1	0.340214608004417
3	58	1	0.293473782962936
4	63	1	0.123494971795201
5	66	0	0.0685969520497134
6	67	0	0.0560051002945604
7	67	0	0.0560051002945604
8	67	0	0.0560051002945604
9	68	0	0.045611454011534
10	69	0	0.037070953178626
12	70	0	0.0300792150451496
14	70	0	0.0300792150451496
11	70	1	0.0300792150451496
13	70	1	0.0300792150451496
15	72	0	0.0197269003820634

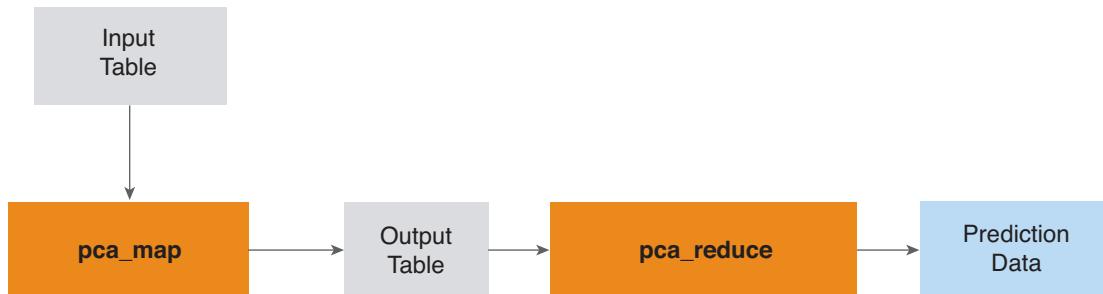
Table 5 - 135: Example output table (continued)

id	temp	damage	fitted_value
16	73	0	0.0159521328508259
17	75	0	0.0104097304438179
18	75	1	0.0104097304438179
19	76	0	0.00840253331401681
20	76	0	0.00840253331401681
21	78	0	0.00546858384315558
22	79	0	0.00440988952099733

Principal Component Analysis (PCA)

Summary

Principal component analysis (PCA) is a common unsupervised learning technique that is useful for both exploratory data analysis and dimensionality reduction. It is often used as the core procedure for factor analysis.



Background

The PCA is a dimension reduction technique. In a lot of cases you may have thousands of input variables. There is a high probability that some of these variables are linearly correlated. Some statistical analysis tools, such as linear regression, do not allow these kind of linearly correlated inputs. Also, high dimensionality causes a lot of problems in the application of statistical tools (see ‘curse of dimensionality’). For all these reasons, it is often desirable to reduce the thousands of potentially linearly correlated input variables to a few linearly uncorrelated variables, which are called principal components. This is what PCA does.

PCA takes an $N \times M$ data matrix (N observations, M variables), and generates an $M \times M$ “rotation matrix.” Each column of the rotation matrix represents an axis in M -dimensional space. The first k columns are the k dimensions along which the data varies most, and thus in some cases can be considered the most important. We can throw away the remaining $M - k$ columns, and we are left with a $M \times k$ rotation matrix. To get the values of our dataset in the

coordinate system of our principal components, we multiply the original $N \times M$ dataset by the $M \times k$ rotation matrix to get a final $N \times k$ matrix. This matrix represents our dataset with a reduced dimensionality of $k \leq M$

As for the results, each eigenvector (output row, less the last standard deviation column) is a weighting scheme over the original input variables, which means the linear combination of the original variables using this eigenvector is a principal component. (Notice that the length of the eigenvector is the same as the number of the original input variables so the multiplication does work.) By choosing the first k eigenvectors, we get k principal components with the k highest standard deviations (due to the eigenvector computation). These principal components are linearly uncorrelated and can be used in further analysis (as input variables).

Note that the rank of principal components decreases in standard deviation, thus significance. Usually the first several principal components would explain 80% - 90% of the total variance, which is sufficient in most applications. That is how ‘dimension reduction’ works.

Usage

Syntax (version 1.1)

```
SELECT * FROM pca_reduce (
    ON pca_map
(
    ON target_table
    [TARGET_COLUMNS(target_columns)]
)
PARTITION BY 1
[COMPONENTS(num_components)]
) ORDER BY component_rank;
```

Arguments

TARGET_COLUMNS	Optional	The columns containing the data. The user can either explicitly list all the names, for example, target_columns('input1', 'input2', ...), or specify a range of columns, for example, target_columns('[4:33]'), or some combination of the above, for example, target_columns('input1', '[4:21]', '[25:53]', 'input73'). Ranges are specified with the syntax: “[<start_column>:<end_column>]”, and the column index starts from 0. These columns must contain numeric values. If this parameter is not specified, the function assumes that every column is part of the data matrix.
COMPONENTS	Optional	The number of principal components to return. If K is specified here, the function will emit the top K components. If this parameter is omitted, the function emits every principle component. This clause must come after the PARTITION BY clause and not before it.

Input Data

See the `TARGET_COLUMNS` parameter, above.

Output

Each row of output represents one eigenvector. The number of eigenvectors output is equal to the number of input columns. The PCA function outputs as its first row the row whose corresponding eigenvalue is the largest eigenvalue in the matrix. The output rows are ranked (`component_rank`) in descending order by the standard deviation of the combination of components along the eigenvector of each row. The `sd` column shows the standard deviation.

Example

Table 5 - 136: Example input table

month	year_1	year_2	year_3	year_4	year_5	year_7	year_10	year_30
Jul-2000	7.05	7.12	7.14	7.15	7.17	7.2	7.24	7.2
Aug-2001	3.73	4.27	4.74	5.06	5.29	5.57	5.8	6.18
Aug-2000	6.95	6.98	6.99	7	7.02	7.04	7.07	7.05
Dec-2001	2.44	3.56	4.33	4.8	5.11	5.5	5.82	6.2
Oct-2001	2.52	3.2	3.8	4.21	4.5	4.9	5.24	5.84
Apr-2001	4.51	4.81	5.12	5.33	5.5	5.75	6	6.41
Mar-2001	4.77	4.95	5.17	5.33	5.46	5.65	5.82	6.14
Nov-2000	6.65	6.58	6.61	6.66	6.7	6.78	6.85	6.91
Jun-2001	4.06	4.63	5.08	5.37	5.58	5.85	6.07	6.41
Nov-2001	2.4	3.2	3.84	4.27	4.57	4.95	5.25	5.74
Sep-2001	3.05	3.69	4.2	4.56	4.82	5.18	5.49	6.04
Dec-2000	6.18	6.06	6.07	6.11	6.14	6.2	6.27	6.41
Jul-2001	3.99	4.58	5.05	5.36	5.57	5.84	6.05	6.38
Jan-2001	5.38	5.44	5.56	5.66	5.74	5.87	6.02	6.26
Oct-2000	6.7	6.65	6.67	6.7	6.73	6.8	6.88	6.94
Feb-2001	5.14	5.28	5.44	5.57	5.68	5.84	6.01	6.29
Sep-2000	6.8	6.79	6.81	6.83	6.86	6.92	7	7.04
May-2001	4.29	4.8	5.19	5.45	5.64	5.9	6.15	6.49

Example SQL-MapReduce call

```
SELECT * FROM pca_reduce (
    ON pca_map
    (
```

```

    ON swap_rates
    TARGET_COLUMNS(' [1:8] ')
)
PARTITION BY 1
) ORDER BY component_rank;

```

Example Output of the PCA Function

Table 5 - 137: Example output table (columns 1 to 5)

component_rank	year_1	year_2	year_3	year_4
1	0.573453730878955	0.463152903493959	0.37942466995694	0.325478612391597
2	-0.621717049638375	-0.149855269326708	0.0983372686076877	0.225158212144758
3	0.316336557227389	-0.116064667091755	-0.256304387077482	-0.297893257430748
4	0.409203418362942	-0.636339801685114	-0.340952832567219	0.104832723271973
5	0.0684982102032094	0.00749860006500552	-0.0220208282305202	-0.120841149766531
6	-0.104101575198423	0.561328630933441	-0.543983136922196	-0.36733888834675
7	0.0324355015122778	0.00565971348028669	-0.137457633913214	-0.272026074962105
8	0.0228313200095594	-0.171625589056941	0.5908711850373	-0.721178571409388

Table 5 - 138: Example output table (columns 6 to 9)

component_rank	year_5	year_7	year_10	year_30
1	0.289238431753085	0.243798827391615	0.208603312959535	0.141875714664729
2	0.30075066956271	0.373537731542053	0.412004661552727	0.361822644787957
3	-0.255917650552815	-0.0624376212391426	0.212423111103479	0.785819295649568
4	0.27709169547578	0.424886062552705	0.00486381117985304	-0.20749422829281
5	-0.348890746381696	-0.0289806878970539	0.826021208059486	-0.418674809962758
6	0.126878166192205	0.463397440255823	-0.0461722076214633	-0.101244202498086
7	0.733571710201333	-0.560626813257105	0.229809801250695	-0.02794095208204
8	0.0875570489092736	0.294492072100244	-0.0658340961772796	-0.0454989370967585

Table 5 - 139: Example output table (column 10)

component_rank	sd
1	2.74055724224795
2	0.357540190302025
3	0.0647262180507067
4	0.0179263021002768

Table 5 - 139: Example output table (column 10)

component_rank	sd
5	0.00784981827414183
6	0.00457726076935829
7	0.00234028096162865
8	0.00207282666031175

Simple Moving Average (stats smavg)

Summary

The Simple Moving Average function computes the average over a number of points in a series.



Background

A 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 the previous 10 days' closing prices.

To calculate this, we compute the arithmetic average of first R rows as specified by the *WINDOW_SIZE* argument. Then, for each subsequent row, compute new value as

$$\text{new_smavg} = \text{old_smavg} - (P_{M-n} + P_M) / N$$

where N is the number of rows as specified by the *WINDOW_SIZE* argument.

Usage

Permissions

You must grant EXECUTE on the function “smavg” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.1)

```
SELECT *
FROM SMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
```

```

        ORDER BY order_by_column
        COLUMNS ('column_names')
        RETURN_ALL('true|false')
        WINDOW_SIZE('window_size')
    ) ;

```

Arguments

<i>COLUMNS</i>	Optional	Specifies the column name for which exponential moving average is required. If this clause is omitted, all the input rows are output as is.
<i>WINDOW_SIZE</i>	Optional	Specifies the number of old values to be considered for calculating the new weighted moving average. Default window_size is 10.
<i>RETURN_ALL</i>	Optional	Specifies whether the first <i>WINDOW_SIZE</i> rows should be output or not. Since the simple moving average for the first <i>WINDOW_SIZE</i> is not defined, nulls will be returned for those columns. Default value is false.

Assumptions

Data is assumed to be partitioned such that each partition contains all the rows of an entity. For example if the simple moving average of a particular share is required, then all transactions of that share should be part of one partition. It is assumed that the input rows are provided in the correct order.

Example

Example Input Data

Table 5 - 140: Example input table: stock_data

symbol	price	volume	ts
APPL	60.33	200	2011-10-04 04:40:00
APPL	59.44	150	2011-10-04 04:41:00
APPL	59.38	200	2011-10-04 04:42:00
APPL	59.38	100	2011-10-04 04:43:00
APPL	59.22	200	2011-10-04 04:44:00
APPL	59.88	300	2011-10-04 04:45:00
APPL	59.55	100	2011-10-04 04:46:00
APPL	59.5	400	2011-10-04 04:47:00
APPL	58.66	410	2011-10-04 04:48:00
APPL	59.05	810	2011-10-04 04:49:00
APPL	57.15	370	2011-10-04 04:50:00
APPL	57.32	470	2011-10-04 04:51:00

Table 5 - 140: Example input table: stock_data (continued)

symbol	price	volume	ts
APPL	57.65	520	2011-10-04 04:52:00
APPL	56.14	120	2011-10-04 04:53:00
APPL	55.33	420	2011-10-04 04:54:00
APPL	55.86	360	2011-10-04 04:55:00
APPL	54.92	3260	2011-10-04 04:56:00
APPL	53.74	1260	2011-10-04 04:57:00
APPL	54.80	160	2011-10-04 04:58:00
APPL	54.86	1650	2011-10-04 04:59:00

Example SQL-MapReduce call

```
SELECT *
  FROM SMAVG
  (
    ON stock_data
    PARTITION BY symbol
    ORDER BY ts
    COLUMNS('price', 'volume')
    WINDOW_SIZE('10')
    RETURN_ALL('true')
  )
 ORDER BY ts;
```

Example Output

Output contains all the input columns, in addition one extra column is output for each of the columns on which simple moving average is requested.

Table 5 - 141: Example output table

symbol	price	volume	ts	price_mavg	volume_mavg
APPL	60.33	200	2011-10-04 04:40:00		
APPL	59.44	150	2011-10-04 04:41:00		
APPL	59.38	200	2011-10-04 04:42:00		
APPL	59.38	100	2011-10-04 04:43:00		
APPL	59.22	200	2011-10-04 04:44:00		
APPL	59.88	300	2011-10-04 04:45:00		
APPL	59.55	100	2011-10-04 04:46:00		
APPL	59.50	400	2011-10-04 04:47:00		
APPL	58.66	410	2011-10-04 04:48:00		
APPL	59.05	810	2011-10-04 04:49:00	59.439	287.0

Table 5 - 141: Example output table (continued)

symbol	price	volume	ts	price_mavg	volume_mavg
APPL	57.15	370	2011-10-04 04:50:00	59.121	304.0
APPL	57.32	470	2011-10-04 04:51:00	58.909000000000006	336.0
APPL	57.65	520	2011-10-04 04:52:00	58.736000000000004	368.0
APPL	56.14	120	2011-10-04 04:53:00	58.412000000000006	370.0
APPL	55.33	420	2011-10-04 04:54:00	58.023	392.0
APPL	55.86	360	2011-10-04 04:55:00	57.621	398.0
APPL	54.92	3260	2011-10-04 04:56:00	57.158	714.0
APPL	53.74	1260	2011-10-04 04:57:00	56.582	800.0
APPL	54.80	160	2011-10-04 04:58:00	56.196	775.0
APPL	54.86	1650	2011-10-04 04:59:00	55.777	859.0



The new columns being added for moving averages are of type real.

Error Messages

You may see this error message:

- ERROR: Moving Average requires the data type of columns to be Double Precision or Integer or BigInteger or Numeric

REASON: One or more columns specified in the COLUMNS arguments are not of correct type.

Weighted Moving Average (stats wmavg)

Summary

The weighted moving average computes the average over a number of points in a time series but applies a weighting to older values. The weighting for the older values decreases arithmetically.



Background

A weighted average is any average that has multiplying factors to 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

the specific meaning of weights that decrease arithmetically. In an n -day WMA, the latest day has weight n , the second latest has $(n - 1)$, and so on, counting down to zero.

```
Total_[M+1] = Total_[M] + P_[M+1] - P_[M-n+1]
Numerator_[M+1] = Numerator_[M] + n*P_[M+1] - Total[M]
new_WMAVG = Numerator_[M+1] / (n(n+1)/2)
```

Where n is the number of rows as specified by the WINDOW_SIZE argument.

Usage

Permissions

You must grant EXECUTE on the function “wmavg” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```
SELECT *
FROM WMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
    COLUMNS('column_names')
    RETURN_ALL('true|false')
    WINDOW_SIZE('window_size')
);
```

Arguments

COLUMNS	Optional	Specifies the column name for which the weighted moving average will be calculated. If this clause is omitted, all the input rows are output as-is.
RETURN_ALL	Optional	Specifies if the first WINDOW_SIZE rows should be output or not. Since exponential moving average for the first WINDOW_SIZE is not defined, nulls will be returned for those columns. Default value is false.
WINDOW_SIZE	Optional	Specifies the number of old values to be considered for calculating the new weighted moving average. Default window_size is 10.

Assumptions

Data is assumed to be partitioned such that each partition contains all the rows of an entity. For example if the exponential moving average of a particular equity share price is required, then all transactions of that equity share should be part of one partition. It is assumed that the input rows are provided in the correct order.

Example

Example Input Data

Table 5 - 142: Example input table: stock_data

symbol	price	volume	ts
APPL	60.33	200	2011-10-04 04:40:00
APPL	59.44	150	2011-10-04 04:41:00
APPL	59.38	200	2011-10-04 04:42:00
APPL	59.38	100	2011-10-04 04:43:00
APPL	59.22	200	2011-10-04 04:44:00
APPL	59.88	300	2011-10-04 04:45:00
APPL	59.55	100	2011-10-04 04:46:00
APPL	59.5	400	2011-10-04 04:47:00
APPL	58.66	410	2011-10-04 04:48:00
APPL	59.05	810	2011-10-04 04:49:00
APPL	57.15	370	2011-10-04 04:50:00
APPL	57.32	470	2011-10-04 04:51:00
APPL	57.65	520	2011-10-04 04:52:00
APPL	56.14	120	2011-10-04 04:53:00
APPL	55.33	420	2011-10-04 04:54:00
APPL	55.86	360	2011-10-04 04:55:00
APPL	54.92	3260	2011-10-04 04:56:00
APPL	53.74	1260	2011-10-04 04:57:00
APPL	54.80	160	2011-10-04 04:58:00
APPL	54.86	1650	2011-10-04 04:59:00

Example SQL-MapReduce call

```
SELECT *
FROM WMAVG
(
    ON stock_data
    PARTITION BY symbol
    ORDER BY ts
    COLUMNS('price','volume')
    WINDOW_SIZE('10')
    RETURN_ALL('true')
)
ORDER BY ts;
```

Example Output

Table 5 - 143: Example output table

symbol	price	volume	ts	price_mavg	volume_mavg
APPL	60.33	200	2011-10-04 04:40:00		
APPL	59.44	150	2011-10-04 04:41:00		
APPL	59.38	200	2011-10-04 04:42:00		
APPL	59.38	100	2011-10-04 04:43:00		
APPL	59.22	200	2011-10-04 04:44:00		
APPL	59.88	300	2011-10-04 04:45:00		
APPL	59.55	100	2011-10-04 04:46:00		
APPL	59.5	400	2011-10-04 04:47:00		
APPL	58.66	410	2011-10-04 04:48:00		
APPL	59.05	810	2011-10-04 04:49:00	59.30072727272727	363.45454545454544
APPL	57.15	370	2011-10-04 04:50:00	58.88454545454545	378.54545454545456
APPL	57.32	470	2011-10-04 04:51:00	58.5570909090909092	408.72727272727275
APPL	57.65	520	2011-10-04 04:52:00	58.32818181818182	442.1818181818182
APPL	56.14	120	2011-10-04 04:53:00	57.856181818181824	397.09090909090907
APPL	55.33	420	2011-10-04 04:54:00	57.29581818181819	406.1818181818182
APPL	55.86	360	2011-10-04 04:55:00	56.90254545454546	400.3636363636364
APPL	54.92	3260	2011-10-04 04:56:00	56.411454545454546	920.7272727272727
APPL	53.74	1260	2011-10-04 04:57:00	55.790000000000006	1020.0
APPL	54.8	160	2011-10-04 04:58:00	55.466	903.6363636363636
APPL	54.86	1650	2011-10-04 04:59:00	55.22309090909091	1062.7272727272727

The output contains the input columns plus a column for each weighted moving average. The weighted moving average output columns have a data type of real.

Error Messages

You may see the following error:

- ERROR: Moving Average requires data type of columns to be Double Precision or Integer or BigInteger or Numeric.

REASON: One or more columns specified in the COLUMNS arguments are not of correct type.

Exponential Moving Average (stats emavg)

Summary

The exponential moving average function, EMAVG, computes the average over a number of points in a time series but applies a damping (weighting) factor to older values. The weighting for the older values decreases exponentially without entirely discarding the older values.

Background

Exponential moving average (EMA), sometimes also called an exponentially weighted moving average (EWMA), applies weighting factors that decrease exponentially. The weighting for each older data point decreases exponentially, giving much more importance to recent observations while still not discarding older observations entirely.

We compute the arithmetic average of the first n rows as specified by `START_ROWS` argument. Then, for each subsequent row, we compute the new value as:

```
new_emavg = alpha * new_value + (1-alpha) * old_emavg
```

Usage

Permissions

You must grant EXECUTE on the function “emavg” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```
SELECT *
FROM EMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
    [COLUMNS('column_names')]
    [RETURN_ALL('true|false')]
    [START_ROWS('number')]
    [ALPHA('alpha_value')]
);
```

Arguments

<code>COLUMNS</code>	Optional	Name of the column name for which the exponential moving average will be calculated. If this clause is omitted, all the input rows are output as-is.
<code>ALPHA</code>	Optional	Specifies the damping factor, which is the degree of weighting decrease. The damping factor must have a value between 0 and 1, which translates to a percentage value of zero to 100. For example, specify an <code>ALPHA</code> of 0.2 to specify a 20% damping factor. A higher <code>ALPHA</code> discounts older observations faster. The default value is 0.1.

<i>START_ROWS</i>	Optional	Specifies the lag, expressed in rows, after which exponential moving average will start to be calculated. The exponential moving average for the first <i>START_ROWS</i> rows is not defined. The default number of <i>START_ROWS</i> is 2.
<i>RETURN_ALL</i>	Optional	Specifies whether the first <i>START_ROWS</i> rows should be included in the output or not. Since exponential moving average for the first <i>START_ROWS</i> is not defined, setting <i>START_ROWS</i> to <i>true</i> causes your query to return nulls for those columns. Default value is false.

Assumptions

This function makes the following assumptions:

- Data is assumed to be partitioned such that each partition contains all the rows of an entity. For example if the exponential moving average of a particular exchange-traded equity share price is required, then all transactions of that equity share should be part of *one partition*.

It is assumed that the input rows are provided in historical order.

Output

For each column on which you calculate an average, the function returns a column containing the moving average. The name of each moving average column is the corresponding input column's name with the suffix, “_mavg” appended to it. All input columns are also returned.

Example

Example Input Data

Table 5 - 144: Example input table: stock_data

symbol	price	volume	ts
APPL	60.33	200	2011-10-04 04:40:00
APPL	59.44	150	2011-10-04 04:41:00
APPL	59.38	200	2011-10-04 04:42:00
APPL	59.38	100	2011-10-04 04:43:00
APPL	59.22	200	2011-10-04 04:44:00
APPL	59.88	300	2011-10-04 04:45:00
APPL	59.55	100	2011-10-04 04:46:00
APPL	59.5	400	2011-10-04 04:47:00
APPL	58.66	410	2011-10-04 04:48:00
APPL	59.05	810	2011-10-04 04:49:00
APPL	57.15	370	2011-10-04 04:50:00
APPL	57.32	470	2011-10-04 04:51:00

Table 5 - 144: Example input table: stock_data (continued)

symbol	price	volume	ts
APPL	57.65	520	2011-10-04 04:52:00
APPL	56.14	120	2011-10-04 04:53:00
APPL	55.33	420	2011-10-04 04:54:00
APPL	55.86	360	2011-10-04 04:55:00
APPL	54.92	3260	2011-10-04 04:56:00
APPL	53.74	1260	2011-10-04 04:57:00
APPL	54.80	160	2011-10-04 04:58:00
APPL	54.86	1650	2011-10-04 04:59:00

Example SQL-MapReduce call

```
SELECT *
FROM EMAVG
(
    ON stock_data
    PARTITION BY symbol
    ORDER BY ts
    COLUMNS('price', 'volume')
    ALPHA('0.1818')
    START_ROWS('10')
    RETURN_ALL('true')
)
ORDER BY ts;
```

Example Output from EMAVG

Table 5 - 145: Example output table

symbol	price	volume	ts	price_mavg	volume_mavg
APPL	60.33	200	2011-10-04 04:40:00		
APPL	59.44	150	2011-10-04 04:41:00		
APPL	59.38	200	2011-10-04 04:42:00		
APPL	59.38	100	2011-10-04 04:43:00		
APPL	59.22	200	2011-10-04 04:44:00		
APPL	59.88	300	2011-10-04 04:45:00		
APPL	59.55	100	2011-10-04 04:46:00		
APPL	59.5	400	2011-10-04 04:47:00		
APPL	58.66	410	2011-10-04 04:48:00		
APPL	59.05	810	2011-10-04 04:49:00	59.439	287.0
APPL	57.15	370	2011-10-04 04:50:00	59.022859800000006	302.0894
APPL	57.32	470	2011-10-04 04:51:00	58.713279888360006	332.61554708000006

Table 5 - 145: Example output table (continued)

symbol	price	volume	ts	price_mavg	volume_mavg
APPL	57.65	520	2011-10-04 04:52:00	58.51997560465616	366.68204062085607
APPL	56.14	120	2011-10-04 04:53:00	58.08729603972967	321.83524563598445
APPL	55.33	420	2011-10-04 04:54:00	57.58601961970682	339.6815979793625
APPL	55.86	360	2011-10-04 04:55:00	57.27222925284413	343.3754834667144
APPL	54.92	3260	2011-10-04 04:56:00	56.84459397467707	873.6178205724657
APPL	53.74	1260	2011-10-04 04:57:00	56.280178790080775	943.8621007923914
APPL	54.8	160	2011-10-04 04:58:00	56.01108228604409	801.3559708683347
APPL	54.86	1650	2011-10-04 04:59:00	55.80181552644128	955.6394553644714

The output consists of a column for each exponential moving average you are calculating, plus all of the input columns returned as-is.



The new columns being added for moving averages are of type real.

Error Messages

You may see the following error message when you run this function:

ERROR: Moving Average requires the data type of columns to be Double Precision or Integer or BigInteger or Numeric.

REASON: One or more columns specified in the COLUMNS arguments are not of correct type.

Volume-Weighted Average Price (stats vwap)

Summary

This function computes, for each in a series of equal-length intervals, the volume-weighted average price of a traded item (usually an equity share). You specify the interval length in the *TIMEINTERVAL* argument.

The first interval starts at the time of the earliest timestamp in the partition, and it ends with the last row timestamped less than *TIMEINTERVAL* seconds later. The second interval starts immediately after the end of the first, and so on. All intervals have the same length.



Background

Compute the sum of the product of the volume and price divided by the total volume traded in a specified window:

```
VWAP = sum(vol*price) / sum(vol)
```

NOTE: In the formula above, "sum" refers to the sum within the current window.

Usage

Permissions

You must grant EXECUTE on the function “vwap” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```
SELECT *
  FROM VWAP
  (
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    ORDER BY date_column
    [PRICE('price_column')]
    [VOLUME ('volume_column')]
    [TIMEINTERVAL('number_of_seconds')]
    [DT('date_column')]
  );
```

Arguments

<i>PRICE</i>	Optional	Name of the traded price column in the input table. Each row typically records one transaction. The <i>PRICE</i> column records the price at which the item traded. Default is "price".
<i>VOLUME</i>	Optional	Name of the column that holds the count of units traded in the transaction(s) recorded in a given row. Default is "volume".
<i>DT</i>	Optional	Name of the column that records the date and time of the trade. Default is "dt"
<i>TIMEINTERVAL</i>	Optional	Specifies the length of the time interval, expressed in seconds. The default is 0, which has the effect of calculating no averages, since every row is considered to be an interval unto itself.

Assumptions

The function has been build with these assumptions:

- Partitioning of input data: The function assumes that you will partition the input data so that each partition contains all the rows of a particular entity. For example, to calculate the volume-weighted average of a particular equity share, then all transactions of that share should be contained within one partition.
- Sort order of input data: The function assumes that the rows are sorted in ascending order based on the *DT* column.
- Start time: The start time for a window is the timestamp of the first row in the window. The timestamp of the first row in the partition is considered to be the start time of the first window. The next window starts immediately after the end of the first, and so on.
- Datatypes: The *DT* input column should be of type `timestamp`. The *PRICE* and *VOLUME* input columns are of type `numeric`, `integer`, `biginteger`, or `real`.

Example Query 1

Input Data for Example 1

The sample table *stockdata* contains these columns:

- `memberid` [int]
- `name` [varchar]
- `dt` [timestamp]
- `price` [numeric]
- `volume` [int]

Table 5 - 146: Example input table: stockdata

memberid	name	dt	price	volume
1	Google	1989-02-20 09:00:45	40	25
1	Google	1989-02-20 09:50:22	50	21
1	Google	1989-02-21 09:00:46	50	29
1	Google	1989-02-22 09:00:46	40	2

Example Query 1

```
SELECT * FROM VWAP (
    ON
    (
        SELECT *
        FROM stockdata
    )
    PARTITION BY memberID
    ORDER BY dt
    PRICE('price')
    VOLUME ('volume')
    DT('dt')
) ;
```

Example 1 Output from VWAP

Table 5 - 147: Example output table

memberid	name	timestamp	vwap
1	Google	1989-02-20 09:00:45	40
1	Google	1989-02-20 09:50:22	50
1	Google	1989-02-21 09:00:46	50
1	Google	1989-02-22 09:00:46	40

All the input columns are output as-is, except for PRICE, VOLUME, and DT, which are not included in the output. In addition, the *timestamp* and *vwap* columns are added to the output table. This example is really an anti-example. Since we omitted the *TIMEINTERVAL* argument in our sample query, the function used the default value of 0, which had the effect of calculating no averages, since every row was considered to be an interval unto itself.

Example Query 2

Input Data for Example 2

Use the same input data you used in Example 1.

Example Query 2

```
SELECT * FROM VWAP
(
ON
(
SELECT *
FROM stockdata
)
PARTITION BY MemberID
ORDER BY dt
PRICE('price')
VOLUME ('volume')
DT('dt')
TIMEINTERVAL ('86400'));
```

Example 2 Output from VWAP

Table 5 - 148: Example output table

MemberID	Name	timestamp	vwap
1	Google	1989-02-20 09:50:22	44.5652
1	Google	1989-02-22 09:00:46	49.3548

This time, since we specified a TIMEINTERVAL of 86,400 seconds (one day), the first two rows are grouped together, and the last two rows are grouped together, and the function calculates the volume-weighted average price for each group.

Error Messages

You may see the following error messages:

- ERROR: Must have column named price or specify name of price column.
REASON: PRICE argument is missing and there exists no column in the input table with the name 'Price'
- ERROR: Must have column named volume or specify name of volume column.
REASON: VOLUME argument is missing and there exists no column in the input table with the name 'Volume'
- ERROR: Must have column named price or specify name of Dt column.
REASON: DT argument is missing and there exists no column in the input table with the name 'Dt'

K-Nearest Neighbor Classification Algorithm (knn)

Summary

The knn function is optimized for small training sets that fit in memory and for large sets. This function supports:

- User-defined distance metrics.

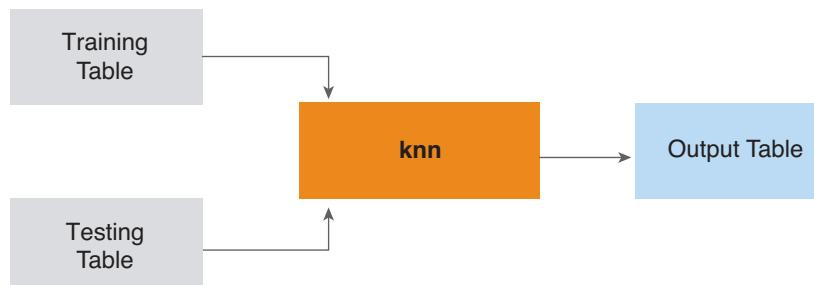
By default, the knn function uses the Euclidean distance metric to determine the proximity to training data objects. However, the knn function lets you define your own distance metric.

- Distance-weighted voting.

The knn function uses distance-weighted voting to help improve the classification accuracy. The knn function calculates distance-weighted voting (w) using this equation:

$$w = 1/\text{pow}(distance, voting_weight)$$

Where *distance* is the distance between the test object and the training object, and *voting_weight* is an argument that you pass to the knn function.



Background

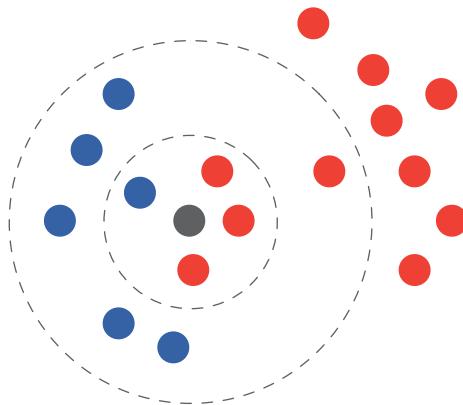
In the IEEE International Conference on Data Mining (ICDM) in December 2006, *k*NN was presented as one of the top 10 ten data mining algorithm. *k*NN is a technique for classifying data objects based on proximity to other data objects with known classification. The object with known classification or labels serve as training data.

*k*NN classifies data based on the following parameters:

- Training data.
- A metric that measures distance between objects.
- The number of nearest neighbors (*k*).

[Figure 7](#) shows an example of how data can be classified using *k*NN. The red dots represent cancerous tissue and the green dots represent normal tissue. If the value of *k* in this example is set to 4 (inner circle), the algorithm classifies the gray dot as cancerous tissue. If *k*=10, the gray dot is classified as normal tissue.

Figure 7: *k*NN Example



Usage

Syntax (version 1.1)

```
SELECT * FROM knn(
    ON (SELECT 1)
    PARTITION BY 1
    TRAINING_TABLE('training_table_name')
    TEST_TABLE('test_table_name')
    NUMBERK(k)
    RESPONSECOLUMN('response_column')
    TEST_POINT_KEY('test_id_column')
    DISTANCE_FEATURES('column1' [, ...])
    [ VOTING_WEIGHT(voting_weight) ]
    [ OUTPUT_TABLE('output_table_name') ]
    [ DOMAIN('host_ip') ]
    [ DATABASE('database_name') ]
    [ USERID('user_name') ]
    PASSWORD('password')
    [CUSTOMIZED_DISTANCE('jar_name', 'distance_class_name')]
    [FORCE_MAPREDUCE('force_mapreduce')]
    [PARTITION_BLOCK_SIZE('partition_Block_Size')] ) ;
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [::1]:2406. The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406).
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
<i>PASSWORD</i>	Required	The Aster Database password of the user running this function.
<i>TRAINING_TABLE</i>	Required	This table holds the training data. Each record in the table represents a classified data object.
<i>TEST_TABLE</i>	Required	This table holds the test data that needs to be classified by the <i>kNN</i> algorithm. Each record represents a test data object.
<i>NUMBERK</i>	Required	The number of nearest neighbors (<i>k</i>) to use for classifying the test data.
<i>RESPONSECOLUMN</i>	Required	This column provides the class label or classification of data objects in the training table.
<i>DISTANCE_FEATURES</i>	Required	The columns in the training table that this function uses to compute the distance between a test object and the training objects. These columns must also be included in the testing table.
<i>TEST_POINT_KEY</i>	Required	The column in the testing table that uniquely identifies an object.
<i>VOTING_WEIGHT</i>	Optional	The voting weight of the distance. The value of this argument must be a non-negative integer. If you do not specify a voting weight, this function uses the default value of 0.
<i>OUTPUT_TABLE</i>	Optional	The name of the output table. The output table contains all the test objects and their classification. If you do not specify an output table, this function sends the output to the console.
<i>CUSTOMIZED_DISTANCE</i>	Optional	Lets you define your own distance function. The parameter <i>jar_name</i> is the name of the jar file containing your distance metric class. The <i>distance_class_name</i> specifies the distance metric class defined in the jar file. The knn function installs the jar file on the Aster Database server. If you do not specify this argument, the knn function uses the Euclidean distance.
<i>FORCE_MAPREDUCE</i>	Optional	Determines whether to partition the training data. If <i>force_mapreduce</i> is true, the knn function partitions the training data and uses the map and reduce function. If this argument is not specified or if <i>force_mapreduce</i> is false, the knn function loads all the training data into memory and only uses the row function.
<i>PARTITION_BLOCK_SIZE</i>	Optional	Denotes the partition block size. This argument is used when <i>force_mapreduce</i> is true. NOTE: It is strongly suggested to set this value according to your training data size and the number of vworkers. If the PARTITION_BLOCK_SIZE value is not suitable or not provided, performance may be not good. As an example, if your training data size is 10 billion and you have 10 workers, the value of PARTITION_BLOCK_SIZE should be 1/N billion, where the value of N is an integer that corresponds to your worker nodes' memory.

Input

The knn function takes two inputs: a training table and a testing table. The columns specified in the *DISTANCE_FEATURES* argument must be present in both the training and testing tables. The columns with the same names in these two tables should contain the same type of data. Also, each row in the testing table presents an object must have a unique identifier.

Output

The output table is a set of predictions for each object in the testing table. The schema of the output table should include a column that for uniquely identifying each object in the table, and another column specifying the classification of the object.

Example

Example Input Tables

In the knn_training table, the ID column uniquely identifies every training object in the table. The Category column specifies the classification of every object.

Table 5 - 149: Example input table: knn_training

ID	X	Y	Z	Category
1	16.21	19.98	8.09	A
...

In the knn_test table, the ID column uniquely identifies every testing object.

Table 5 - 150: Example input table: knn_test

ID	X	Y	Z
1	9.0	10.2	6.09
...

Example SQL-MR Call

```
SELECT * FROM knn(
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    TRAINING_TABLE('knn_training')
    TEST_TABLE('knn_test')
    NUMBERK(3)
    RESPONSECOLUMN('category')
    DISTANCE_FEATURES('x', 'y')
    VOTING_WEIGHT(1)
    TEST_POINT_KEY('id')
    DISTANCE('a.jar', 'com.example.MyDistance')
    OUTPUT_TABLE('knn_output')
);
```

Example Output

After running this example, the function sends this message to the Console.

Successful!

The final result are stored in the table knn_output

In addition, the function generates the knn_output.

Table 5 - 151: knn_output

id	category
1	A
2	B
...	...

User-Defined Distance Metric

A user-defined distance metric is used in this example. The Java class com.example.MyDistance defines this metric:

```
package com.example;

import com.asterdata.ncluster.sqlmr.data.RowView;
import com.asterdata.sqlmr.analytics.classification.knn.distance.Distance;

public class MyDistance implements Distance {

    /**
     * calculate the distance between the test row and the training row.
     * note: 1. don't reverse the sequence of parameters
     *       2. the columns of trainingRowView is 'responseColumn, f1,f2,...,fn'
     *       3. the columns of testRowView is the same as TEST_TABLE
     *       4. all the trainingRowView and testRowView is zero-based
     *          (0 <= index && index < getColumnCount())
     *
     * @param testRowView
     *         stands for a point in the test data set
     * @param trainingRowView
     *         stands for a point in the training data set, the columns is the
     *         columns in distanceFeatures argument
     * @return the double value of distance
    */
    @Override
    public double calculate(RowView testRowView, RowView trainingRowView) {
        return Math.abs(testRowView.getIntAt(1) - trainingRowView.getIntAt(1));
    }
}
```

Error Messages

When using this function, you may encounter the following error messages:

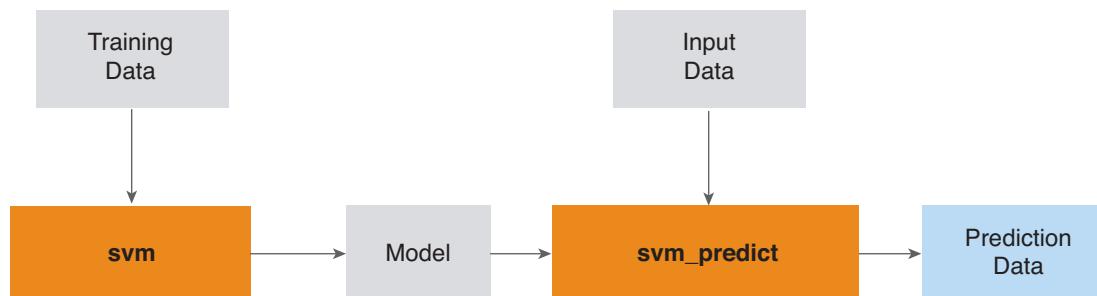
- ERROR: numberK must greater than 0
REASON: The value of the *NUMBERK* argument is 0. You must provide a value greater than 0.
- ERROR: votingWeight must not less than 0
REASON: The value of the *VOTING_WEIGHT* argument is 0. You must provide a value greater than 0.
- ERROR: the column xxx doesn't exist in table yyy
REASON: The specified column is missing from the specified table.
- ERROR: The table 'output_table' has existed in the database
REASON: The output table already exists in the database.

Support Vector Machines

Summary

SQL-MR provides two functions that implement the Support Vector Machines (SVM) algorithm, which is among the most popular “out of the box” classification algorithms:

- *svm*—Takes as input a table containing training data and uses it to generate a model that can be used by the *svm_predict* function to predict data classification.
- *svm_predict*—Uses the model generated by the *svm* function to predict the classification of documents in the input table. For example, you can use this function to determine whether the input news stories are about the stock market or some other type of news.



The SVM algorithm is a binary classification algorithm that can be adapted to support multiple classes using the *machine learning reductions* technique. The SVM algorithm is extremely popular because it has relatively few tunable parameters, and performs very well compared to other algorithms such as Naive Bayes or logistic regression.

Background

The objective of the SVM algorithm is similar to logistic regression: given a set of predictor variables, classify an object as one of two possible outcomes. However, unlike logistic regression, the methods that the SVM algorithm uses to classify objects are very different.

Intuitively, logistic regression develops a probabilistic model based on the input data, and given a test instance x , estimates the probability that x belongs to a particular class.

In contrast, the SVM algorithm ignores the probabilistic interpretation. Instead, it attempts to find the boundary that maximizes the distance between two classes. To compute a class prediction given a test instance x , the SVM algorithm determines on which side of the boundary x lies.

Because the SVM algorithm is a binary classification algorithm, it can achieve multinomial classifications using machine learning reductions, such as one-against-all or pairwise classifications. In a K -class classification problem, the SVM algorithm trains the K support vector machines. The algorithm labels the n^{th} class as 1, and all other classes as 0. In the test phase, each of the SVM algorithms trains each test observation using each of the K support vector machines. The class that results in the most observations predicted as 1 is the resulting prediction.

Implementation Notes

This implementation solves the primal form of a linear kernel support vector machine via gradient descent on the objective function. It is based primarily on *Pegasos: Primal Estimated Sub-Gradient SOLver for SVM* (by S. Shalev-Shwartz, Y. Singer, and N. Srebro; presented at ICML 2007).

Currently, the `svm` SQL-MR function uses some approximation and only linear kernel is available.

svm

This function takes as input a set of training data and generates a model that can be used by the `svm_predict` function to classify data. This function operates on sparse vector style input data, where one partition represents one observation.



This document assumes that you have the ability to create training data that can be used by the `svm` function.

The `svm` function generates `vec_index` and `vec_value` (see the syntax below) as columns. The GROUP BY operation around the SQL-MR function aggregates each of the local models into a global one. After building the model using the `svm` function, you need to export it, and reload it with the `\install` command. For example:

```
\o svm_model.csv
COPY output_table TO STDOUT WITH DELIMITER ',';
\o
\install svm_model.csv
```

Usage

Syntax (version 1.0)

```
CREATE TABLE output_table (PARTITION KEY(vec_index) ) AS
SELECT vec_index, avg(vec_value) as vec_value
FROM svm(
    ON input_table
    PARTITION BY id_column
    OUTCOME( outcome_column )
    ATTRIBUTE_NAME( attr_name_column )
    ATTRIBUTE_VALUE( attr_value_column )
)
GROUP BY vec_index;
```

Parameters

<i>input_table</i>	Required	The name of the input table containing the training data.
<i>OUTCOME</i>	Required	The name of the column containing the classification of the tokens in the training data.
<i>ATTRIBUTE_NAME</i>	Required	The name of the column containing the input tokens.
<i>ATTRIBUTE_VALUE</i>	Required	The name of the column containing the scores of the input tokens.
<i>output_table</i>	Required	The name of the output table containing the prediction model.

Example

Input

In this example, the input table ([Table 5 - 152](#)) containing the classified data is partitioned by doc_id. Each partition represents a single document. Each row contains a document ID, the name of a token contained in the document, a tf-idf score associated with that token in the document, and the category of the document (a binary outcome, either -1 or 1).

Table 5 - 152: Input Model to the svm Function

doc_id	token	tf_idf_score	category
14	quick	0.18	-1
14	brown	0.03	-1
14	fox	0.46	-1
14	jumped	0.29	-1
14	over	0.21	-1
14	lazy	0.44	-1
14	dog	0.33	-1
33	start	0.11	1
33	test	0.18	1
33	vm	0.23	1
33	diagnostic	0.44	1

Example SQL-MR Call

```
CREATE TABLE sample_model (PARTITION KEY(vec_index)) AS
SELECT vec_index, avg(vec_value) as vec_value
FROM svm(
    ON sample_table
    PARTITION BY doc_id
    OUTCOME( 'category' )
    ATTRIBUTE_NAME( 'token' )
    ATTRIBUTE_VALUE( 'tf_idf_score' )
)
GROUP BY vec_index;
```

Output

In this example, the `svm` function creates an SVM model and stores it in the `sample_model` table. Export this model and reload it into your Aster Database:

```
\o sample_model.csv
COPY sample_model TO STDOUT WITH DELIMITER ',';
\o
\install sample_model.csv
```

svm_predict

This function takes as input the model generated by the `svm` function and uses the model to make classification predictions.

Usage

Syntax (version 1.0)

```
SELECT * FROM svm_predict(
    ON input_table
    PARTITION BY id_column
    ATTRIBUTE_NAME(attr_name_column)
    ATTRIBUTE_VALUE(attr_value_column)
    THRESHOLD('threshold_value1', ... , 'threshold_valueN')
    WEIGHT_FILE(weight_file_name)
);
```

Parameters

<i>input_table</i>	Required	The name of the input table containing the data to be classified.
<i>ATTRIBUTE_NAME</i>	Required	The column containing the names of the input tokens.
<i>ATTRIBUTE_VALUE</i>	Required	The column containing the scores of the input tokens.
<i>THRESHOLD</i>	Optional	The double-valued threshold. Classifier output values above this threshold are predicted as the positive case, while values below are predicted as the negative case. A user can enter multiple values for this parameter, which results in one prediction for each threshold value specified.
<i>WEIGHT_FILE</i>	Required	The name of the installed CSV file containing the SVM model.



Ideally, you should not need to set the threshold parameter. However, in some cases, to get the best performance from the model, you need to experiment with this parameter. You do so by running the function with thresholds from -1 to 1 in 0.1 increments. After running the function, select the threshold parameter that performs best on either a hold-out set of test data or a cross-validation procedure.

Examples

Example 1

Input

To score rows using the model generated by svm, use the svm_predict SQL-MR function. The format of the input table to the svm_predict function is the same as the format of the table containing the training data, except that the last column in the training table is not part of the input table. When calling the svm_predict function, you can specify a list of thresholds. The binary predictions (-1/1) are based on the thresholds you provide.

Table 5 - 152 lists the contents of the input table, svm_predict_input_data.

Table 5 - 153: Example input table: svm_predict_input_data

doc_id	token	tf_idf_score
14	quick	0.18
14	brown	0.03
14	fox	0.46
14	jumped	0.29
14	over	0.21
14	lazy	0.44
14	dog	0.33
33	start	0.11
33	test	0.18
33	vm	0.23
33	diagnostic	0.44

Example SQL-MR call

```
SELECT * FROM svm_predict(
    ON svm_predict_input_data
    PARTITION BY doc_id
    ATTRIBUTE_NAME( 'token' )
    ATTRIBUTE_VALUE( 'tf_idf_score' )
    THRESHOLD( '0' )
    WEIGHT_FILE( 'sample_model.csv' ))
ORDER BY item_id;
```

Example output

In this example, the `svm_predict` function generates this table:

Table 5 - 154: Example output table

item_id	raw_prediction	prediction_0.0
14	-8.21948909102412	-1
33	5.39444160469809	1

Example 2—A numeric classification example

Example input

This example is from the 'iris' data in R. The original data has one response variable 'Species', and four attributes: 'Sepal.Length', 'Sepal.Width', 'Petal.Length', and 'Petal.Width'.

To use SVM, you should prepare the data as follows:

Table 5 - 155: Input data

id	attribute	value	category
1	slength	5.1	-1
1	swidth	3.5	-1
1	plength	1.4	-1
1	pwidth	0.2	-1
2	slength	4.9	-1
2	swidth	3.0	-1
2	plength	1.4	-1
2	pwidth	0.2	-1
3	slength	4.7	-1
3	swidth	3.2	-1
3	plength	1.3	-1
3	pwidth	0.2	-1
4	slength	4.6	-1
4	swidth	3.1	-1
4	plength	1.5	-1
4	pwidth	0.2	-1
5	slength	5.0	-1
5	swidth	3.6	-1
5	plength	1.4	-1

Table 5 - 155: Input data

id	attribute	value	category
5	pwidth	0.2	-1
6	slength	5.4	-1
6	swidth	3.9	-1
6	plength	1.7	-1
6	pwidth	0.4	-1
7	slength	4.6	-1
7	swidth	3.4	-1
7	plength	1.4	-1
7	pwidth	0.3	-1
8	slength	5.0	-1
8	swidth	3.4	-1
8	plength	1.5	-1
8	pwidth	0.2	-1
9	slength	4.4	-1
9	swidth	2.9	-1
9	plength	1.4	-1
9	pwidth	0.2	-1
10	slength	4.9	-1
10	swidth	3.1	-1
10	plength	1.5	-1
10	pwidth	0.1	-1
11	slength	7.0	1
11	swidth	3.2	1
11	plength	4.7	1
11	pwidth	1.4	1
12	slength	6.4	1
12	swidth	3.2	1
12	plength	4.5	1
12	pwidth	1.5	1
13	slength	6.9	1

Table 5 - 155: Input data

id	attribute	value	category
13	swidth	3.1	1
13	plength	4.9	1
13	pwidth	1.5	1
14	slength	5.5	1
14	swidth	2.3	1
14	plength	4.0	1
14	pwidth	1.3	1
15	slength	6.5	1
15	swidth	2.8	1
15	plength	4.6	1
15	pwidth	1.5	1
16	slength	5.7	1
16	swidth	2.8	1
16	plength	4.5	1
16	pwidth	1.3	1
17	slength	6.3	1
17	swidth	3.3	1
17	plength	4.7	1
17	pwidth	1.6	1
18	slength	4.9	1
18	swidth	2.4	1
18	plength	3.3	1
18	pwidth	1.0	1
19	slength	6.6	1
19	swidth	2.9	1
19	plength	4.6	1
19	pwidth	1.3	1
20	slength	5.2	1
20	swidth	2.7	1
20	plength	3.9	1

Table 5 - 155: Input data

id	attribute	value	category
20	pwidth	1.4	1

Example SQL-MR call—training

```
CREATE TABLE iris_sample_model (PARTITION KEY(vec_index) ) AS
SELECT vec_index, avg(vec_value) as vec_value FROM
svm(
    ON iris_sample
    PARTITION BY id
    OUTCOME( 'category' )
    ATTRIBUTE_NAME( 'attribute' )
    ATTRIBUTE_VALUE( 'value' )
) GROUP BY vec_index;
```

This creates an SVM model in the table iris_sample_model. Export the model and reload it into Aster Database as follows:

```
\o iris_sample_model.csv
COPY iris_sample_model TO STDOUT WITH DELIMITER ',';
\o
\install iris_sample_model.csv
```

To score rows using this model, use the svm_predict() function. The input format of the test rows is exactly the same as the training rows, except that the OUTCOME column is omitted. You can specify a list of thresholds. The binary predictions (-1/1) are based on the thresholds you provide. This example uses the same input table as test data to perform the predictions (fitting). As evident in the output, by choosing the appropriate threshold, the fitting accuracy is high.

```
SELECT * FROM svm_predict(
    ON iris_sample
    PARTITION BY id
    ATTRIBUTE_NAME( 'attribute' )
    ATTRIBUTE_VALUE( 'value' )
    THRESHOLD( '63.7' )
    WEIGHT_FILE( 'iris_sample_model.csv' )
) ORDER BY item_id;
```

Example output table

Table 5 - 156: Example output table

item_id	raw_prediction	prediction_63.7
1	58.0414180818017	-1
2	54.7389367236685	-1
3	53.42287849253	-1
4	53.3576265299552	-1
5	57.6453995242311	-1
6	63.6442320388121	-1
7	54.0756414850731	-1
8	57.4521486016999	-1
9	50.6070324520134	-1
10	55.4554620812542	-1
11	90.5900717493738	1
12	85.1802411240276	1
13	90.6909487358374	1
14	72.2082931951648	1
15	85.0243717767284	1
16	78.1308657915167	1
17	85.9983857619405	1
18	63.8724949841645	1
19	85.8054050850991	1
20	71.0209802334234	1

ConfusionMatrix

Summary

In the field of artificial intelligence, a confusion matrix is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix).

Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class.

The name stems from the fact that it makes it easy to see if the system is confusing two classes (that is, commonly mislabeling one as another). Outside artificial intelligence, the confusion matrix is often called the contingency table or the error matrix.



Usage

Syntax (version 1.0)

```
select * from ConfusionMatrix(
    ON {table_name|view_name|(query)} PARTITION BY <expect_column_name>
    EXPECTCOLUMN('expect_column_name')
    PREDICTCOLUMN('result_column_name')
) ;
```

Arguments

<i>EXPECTCOLUMN</i>	Required	The name of the column with the expected category. Note: Keep the value of this clause consistent with the value of "PARTITION BY"; otherwise the output will be not correct.
<i>PREDICTCOLUMN</i>	Required	The name of the column with the predicted category.

Output

The output table is a N×4 matrix, where
 $N = \text{distinct}(\text{expected categories}) * \text{distinct}(\text{predicted categories})$

The 4 columns in the output schema are:

Table 5 - 157: Output schema

Column	Descriptions
expect	The expected category.
predict	The predicted category.
count	How many predictions of an expected category relative to a predicted category.

Table 5 - 157: Output schema (continued)

Column	Descriptions
fraction	<p>The value of fraction can be calculated by this equation:</p> $\text{fraction}(\text{expect}, \text{predict}) = \text{count}(\text{expect}, \text{predict}) / (\text{count}(\text{expect}, \text{predict1}) + \text{count}(\text{expect}, \text{predict2}) + \dots + \text{count}(\text{expect}, \text{predictm}))$ <p>where m is the count of the distinct values of predicted categories.</p>

For example, If the distinct values of expectColumn is *c1*, *c2*, and *c3*, the distinct values of predictColumn is *c1* and *c3*. The output looks like:

Table 5 - 158: Output example

expect	predict	count	fraction
c1	c1	3	0.75
c1	c3	1	0.25
c2	c1	2	0.5
c2	c3	2	0.5
c3	c1	1	0.25
c3	c3	3	0.75

Example

Input

Table 5 - 159: Input table

id	expect	predict
1	dog	dog
2	dog	dog
3	dog	cat
4	dog	dog
5	cat	cat
6	cat	cat
7	cat	cat
8	cat	rat
9	rat	dog
10	rat	cat
11	rat	cat
12	rat	rat
13	rat	rat

SQL-MapReduce Call

```
select * from ConfusionMatrix(
    on test_table
    PARTITION BY expect
    expectcolumn('expect')
    predictcolumn('predict')
) ;
```

Output

Table 5 - 160: Output table

expect	predict	count	fraction
dog	dog	3	0.75
dog	cat	1	0.25
cat	cat	3	0.75
cat	rat	1	0.25
rat	dog	1	0.2
rat	cat	2	0.4
rat	rat	2	0.4

Percentile

Summary

This analytical function can be used to find percentiles on a per group basis.



Background

This function generates the exact percentiles for a group of numbers. This function works well when the number of groups is big (in the order of > 100s of thousands) and each group is small enough to fit in memory (in the order of 10s of thousands). The exact number depends on the cluster configuration.

Usage

Usage

```
SELECT * FROM percentile(
    ON <table>
    PARTITION BY <PARTITION BY col> [, <PARTITION BY col>]* *
    PERCENTILE('<percentile>' [, '<percentile>']*)
    TARGET_COLUMNS('<column_name>' [, '<column_name>']*)
    [GROUP_COLUMNS('<column_name>' [, '<column_name>']*) ]
) ;
```

Parameters

PERCENTILE	Required	The comma separated list of percentile values required.
TARGET_COLUMNS	Required	The comma separated list of column names for which percentile values need to be retrieved.
GROUP_COLUMNS	Optional	The comma separated list of column names for which needs to be passed along. Generally, this list is same as the list of columns specified in the partition by clause.

Example

Input

Table 5 - 161: Input table

name	country	age	height	weight	sex	sport
Rasul Abduraim	Kyrgyzstan	23	180	null	M	Taekwondo
Georgios Achilleos	Cyprus	31	172	80	M	Shooting
Janet Achola	Uganda	24	167	52	F	Athletics
Abrar Osman Adem	Eritrea	18	168	58	M	Athletics
...

Example SQL-MR Call

```
select * from percentile
  (on london_olympics
   partition by country
   percentile('0','100')
   target_columns('height', 'weight', 'age')
   group_columns('country')
  ) ;
```

Output

Table 5 - 162: Output Table

country	percentile	height	weight	age
Cyprus	0	162	55	16
Cyprus	100	193	110	37
El Salvador	0	159	54	19
El Salvador	100	185	80	29
Eritrea	0	160	51	18
Eritrea	100	188	71	34
Guatemala	0	147	44	16
Guatemala	100	190	132	41

Table 5 - 162: Output Table (continued)

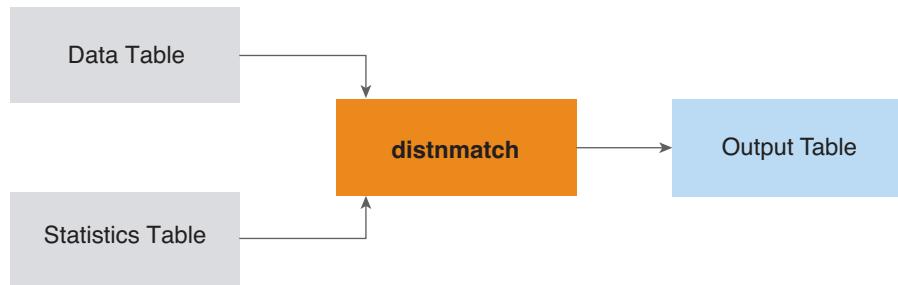
country	percentile	height	weight	age
Indonesia	0	152	49	16
Indonesia	100	183	80	30
Kyrgyzstan	0	155	52	16
Kyrgyzstan	100	186	100	29
Puerto Rico	0	170	48	16
Puerto Rico	100	191	86	32
Saudi Arabia	0		59	16
Saudi Arabia	100		94	42
Uganda	0	153	50	15
Uganda	100	183	76	33
Albania	0	160	51	16
Albania	100	194	130	44
Grenada	0	162	56	19
Grenada	100	187	75	32
Kuwait	0	165	55	17
Kuwait	100	195	110	48
Peru	0	155	46	17
Peru	100	183	80	33
Philippines	0	157	53	18
Philippines	100	180	125	33
Qatar	0	155	43	17
Qatar	100	189	104	41
Tajikistan	0	160	55	19
Tajikistan	100	189	120	44
Turkmenistan	0	165	56	18
Turkmenistan	100	194	100	30
Vietnam	0	150	45	15
Vietnam	100	178	80	37
Bahrain	0	162	43	15
Bahrain	100	175	72	30

Table 5 - 162: Output Table (continued)

country	percentile	height	weight	age
Costa Rica	0	163	50	21
Costa Rica	100	195	96	32
Lebanon	0	152	52	18
Lebanon	100	188	87	26
Luxembourg	0	157	52	20
Luxembourg	100	192	80	49
Mauritius	0	154	52	18
Mauritius	100	187	75	32

Distribution Matching

The distribution matching function `distnmatch` carries out hypothesis testing and finds the best matching distribution for the data



- `distnmatch` (“Hypothesis Test” Mode)
- `distnmatch` (“Best Match” Mode)

distnmatch (“Hypothesis Test” Mode)

Summary

The `distnmatch` function tests the hypothesis that the sample data comes from the specified reference distribution. In this mode, this function carries out up to four tests simultaneously:

- Anderson-Darling test
- Kolmogorov-Smirnov test
- Cramér-von Mises criterion
- Pearson’s Chi-squared test

Usage

Syntax (Continuous Distributions) (version 1.0)

Use the first option when processing large data groups and each group's data might be stored across multiple nodes.

Use the second option when processing small data groups and each group's data can be stored on a single node.



Ultimately, performance depends on the data you are processing. You might want to try both options using sample data to decide which syntax option works best.

Option 1: FACT + DIMENSION

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER (PARTITION BY col1, col2, ..., colN
            ORDER BY column_name) AS rank, *
            FROM input_table WHERE column_name IS NOT NULL)
        AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size FROM input_table WHERE
            column_name IS NOT NULL GROUP BY col1, col2,..., colN)
        AS groupstats DIMENSION
        VALUECOLUMN(column_name)
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Option 2: CO-GROUP

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER (PARTITION BY col1, col2, ..., colN
            ORDER BY column_name) AS rank, * FROM input_table
            WHERE column_name IS NOT NULL) AS input PARTITION BY
            col1, col2, ..., colN
        ON (SELECT col1, col2, ..., colN, COUNT(*) AS
            group_size FROM input_table WHERE
            column_name IS NOT NULL GROUP BY col1, col2, ..., colN) AS
            groupstats PARTITION BY col1, col2, ..., colN
        VALUECOLUMN( column_name )
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Syntax (Discrete Distributions) (version 1.0)

Option 1: FACT + DIMENSION

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1)) OVER
            (PARTITION BY col1, col2, ..., colN ORDER BY
                column_name) AS rank, col1, col2, ..., colN, column_name
            FROM input_table WHERE column_name IS NOT NULL GROUP BY col1,
            col2, ..., colN, column_name) AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN, COUNT(*) AS
            group_size FROM input_table WHERE column_name IS NOT NULL
            GROUP BY col1, col2, ..., colN) AS groupstats DIMENSION
        VALUECOLUMN( column_name )
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Option 2: CO-GROUP w/ ORDER BY

Use this format if you want to run the CvM test on discrete distributions; otherwise the results might be incorrect.

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1)) OVER (PARTITION BY
            col1, col2, ..., colN ORDER BY column_name) AS rank, col1,
            col2, ..., colN, column_name FROM input_table WHERE
            column_name IS NOT NULL GROUP BY col1, col2, ..., colN,
            column_name) AS input PARTITION BY col1, col2, ..., colN
            ORDER BY column_name
        ON (SELECT col1, col2, ..., colN, COUNT(*) AS group_size FROM
            input_table WHERE column_name IS NOT NULL GROUP BY col1,
            col2, ..., colN) AS groupstats PARTITION BY col1, col2,
            ...
            colN
        VALUECOLUMN( column_name )
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Arguments

VALUECOLUMN	Required	Column containing the data. For continuous distributions, the data type can be any numeric type. For discrete distributions, data type must be integer type.
TESTS	Optional	A list of tests the user wants to perform. Supported tests are 'KS' (Kolmogorov-Smirnov test), 'CvM' (Cramér-von Mises criterion), 'AD' (Anderson-Darling test), 'CHISQ' (Pearson's Chi-squared test). By default, all tests are run.

<i>DISTRIBUTIONS</i>	<i>Required</i>	A list of reference distributions (must be continuous distributions) to test the data against. Must be one of (continuous distributions) BETA, CAUCHY, CHISQ, EXPONENTIAL, F, GAMMA, LOGNORMAL, NORMAL, T, TRIANGULAR, UNIFORMCONTINUOUS, and WEIBULL, or (discrete distributions) BINOMIAL, GEOMETRIC, NEGATIVEBINOMIAL, POISSON, and UNIFORMDISCRETE. When specifying a reference distribution, you must also define parameters associated with that distribution. For example, when using a NORMAL distribution, you can set the MEAN and SD of the distribution. When using a CHISQ distribution, you must set the degrees of freedom. Here are the format that how you should provide parameter(s) for each distribution:
<i>GROUPINGCOLUMN</i>	<i>Optional</i>	The column containing the group identifications over which the test will run. The function can run multiple tests for different partitions of the data in parallel. If there is no grouping column, then PARTITION BY 1 should be used. In the second ON clause, <i>col1</i> , <i>col2</i> , ..., <i>colN</i> and GROUP BY should be omitted.
<i>MINGROUPSIZE</i>	<i>Optional</i>	The minimum size of a group. Any group containing less points than this value is not considered while running the statistical tests. Default value is 50.
<i>CELLSIZE</i>	<i>Optional</i>	The number of cells that you want to discretize the continuous distribution. The default value is 10. Note that <i>MINGROUPSIZE/CELLSIZE</i> can not be less than 5, otherwise an error will be raised. Also, <i>CELLSIZE</i> must be greater than 3 if <i>NORMAL</i> distribution is tested against, and greater than 1 in all other situations.

Notes:

- For Normal distribution, the sample mean and standard deviation should be used as parameters.
- For other continuous and discrete distributions, the parameters are treated as pre-specified and not estimated from the data.

Inputs

This function expects two tables.

- [Table 1](#)
- [Table 2](#)

Table 1

The first table (Table 1) contains the data, upon which the matching is run, in the value column. The data is first sorted (globally or within each group), and a rank is appended to each row. If you want to avoid sorting inside the database, provide the input table with a ‘rank’ column, and write use this clause instead:

```
ON SELECT * FROM input_table
```

For continuous distributions, table 1 should have the following schema:

Table 5 - 163: Table 1 schema for continuous distributions

Column	Description
rank	For each value.
All input columns in the original order	Contain the value column as well as the grouping columns.

For discrete distributions, table 1 should have the following schema:

Table 5 - 164: Table 1 schema for discrete distributions

Column	Description
counts	For each distinct integer.
rank	Accumulated counts order by value.
grouping columns	
value column	distinct integers

Table 2

The second table (Table 2) summarizes the counts for each group. If there is no grouping column, then omit the 'GROUP BY ...' clause.

Output

For each group (g_k) and each reference distribution ($distn_i$), the output has the following schema:

Table 5 - 165: Output schema

Column Name
grouping columns
group_size
...
distn _i _KS_statistic
distn _i _KS_p-value
distn _i _CvM_statistic
distn _i _CvM_p-value
distn _i _AD_statistic
distn _i _AD_p-value
distn _i _CHISQ_statistic

Table 5 - 165: Output schema (continued)

Column Name
distrn_i_CHISQ_p-value
...

Results (statistics and p-values):

- For a normal distribution: p-values are computed from the modified statistics according to [nortest](#).
- For other continuous distributions:
 - KS: p-values are computed using approximation to the Kolmogorov-Smirnov distribution (assuming large sample size), and results are comparable to the ks.test() from stats package in R.
 - CvM: no good suggestions, and results are computed in the same way as [nortest](#) for now.
 - AD: results are comparable to the ad.test() from [ADGoFTest](#) package in R.
 - CHISQ: statistics are comparable to pearson.test() from [nortest](#) package in R with a minor modification (when data falls beyond the upper limit of the distribution's support), and p-values are computed using DOF = CELLSIZE - 1.
- For discrete distributions:
 - KS: results are comparable to the ks.test() from the [dgof](#) package in R when reference distribution is a 'stepfun' object.
 - CvM: statistics are comparable to the cvm.test() from the [dgof](#) package in R when reference distribution is a 'stepfun' object, and p-values are not calculated due to complexity.
 - AD: same as 'other continuous distributions'.
 - CHISQ: same as 'other continuous distributions'.

Examples



Tip: Before running the examples in this section, switch the output mode in Act to *expanded output* by entering `\x` at the Act command prompt. With *expanded output* mode turned on, each record is split into rows, with one row for each value, and each new record is introduced with a text label in a form like: ---[RECORD 37]---. This mode helps make wide tables readable on a small screen, and is very useful if you are trying to read EXPLAIN output.

Example 1—Normality Tests with 'groupingColumns'

In this example, run all of the four tests ('KS', 'CvM', 'AD', 'CHISQ') on a sample generated by a Normal distribution with mean = 50 and sd = 2.

Input

Here is a snapshot of the input data:

Table 5 - 166: Input data

times	storeid	variable_name	price
2011-11-11 11:11:11	1	variable1	44.6267
2012-12-12 12:12:12	2	variable2	44.6267
2011-11-11 11:11:11	1	variable1	44.9272
2012-12-12 12:12:12	2	variable2	44.9272
2011-11-11 11:11:11	1	variable1	45.1384
2012-12-12 12:12:12	2	variable2	45.1384
...

Example SQL-MR Call (FACT + DIMENSION)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT RANK() OVER (PARTITION BY times, storeid, variable_name
        ORDER BY price) AS rank, *
            FROM distnmatch_normal_50_2 WHERE price IS NOT NULL) AS input
        PARTITION BY ANY
        ON (SELECT times, storeid, variable_name, COUNT(*) AS group_size FROM
            distnmatch_normal_50_2 WHERE price IS NOT NULL GROUP BY times,
            storeid, variable_name) AS groupstats DIMENSION
        VALUECOLUMN('price')
        TESTS('KS', 'CvM', 'AD', 'CHISQ')
        DISTRIBUTIONS('NORMAL:49.97225,2.009698')
        GROUPINGCOLUMNS('times', 'storeid', 'variable_name')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    ) PARTITION BY times, storeid, variable_name);
```

The parameters '49.97225,2.009698' provided here are the sample mean and standard deviation of the data, respectively.

Output

The following tables show two rows in expanded mode. The first column entries are output column names. The second column entries are corresponding values. The only difference here is the first three grouping columns.

Table 5 - 167: Output table

Column Name	Value
-[RECORD 1]-----+	
times	2011-11-11 11:11:11
storeid	1
variable_name	variable1
group_size	400
NORMAL:49.97225,2.009698_KS_statistic	0.0319503
NORMAL:49.97225,2.009698_KS_p-value	0.41181
NORMAL:49.97225,2.009698_CvM_statistic	0.0556535
NORMAL:49.97225,2.009698_CvM_p-value	0.430792
NORMAL:49.97225,2.009698_AD_statistic	0.376151
NORMAL:49.97225,2.009698_AD_p-value	0.410292
NORMAL:49.97225,2.009698_CHISQ_statistic	7.8
NORMAL:49.97225,2.009698_CHISQ_p-value	0.35056
-[RECORD 2]-----+	
times	2012-12-12 12:12:12
storeid	2
variable_name	variable2
group_size	400
NORMAL:49.97225,2.009698_KS_statistic	0.0319503
NORMAL:49.97225,2.009698_KS_p-value	0.41181
NORMAL:49.97225,2.009698_CvM_statistic	0.0556535
NORMAL:49.97225,2.009698_CvM_p-value	0.430792
NORMAL:49.97225,2.009698_AD_statistic	0.376151
NORMAL:49.97225,2.009698_AD_p-value	0.410292
NORMAL:49.97225,2.009698_CHISQ_statistic	7.8
NORMAL:49.97225,2.009698_CHISQ_p-value	0.35056

Compare with R

For comparison, you can use the package nortest in R as a benchmark. It contains five tests for normality, four of which are discussed above. For each test 'f', you only need to provide the data 'x'. Then, f(x) returns the test statistic and p-value. All the tests use the sample mean and standard deviation as parameters of the null Normal distribution.

```
# Convert the table into a file, then read the data from file
x = read.table(file='raw_normal_50_2.txt')

# Install the package if you have not downloaded it yet.
install.packages('nortest', dependencies=FALSE)

# Load the package
library(nortest)

# Run all the normality tests
lillie.test(x)
cvm.test(x)
ad.test(x)
pearson.test(x, n.classes = 10)
```

Here are the tests results. Note that other packages might use different methods for computing p-values, but the test statistics should be the same. For normality tests in distnmatch, the p-values are computed from the modified statistics according to nortest.

```
# results
> lillie.test(x)
  Lilliefors (Kolmogorov-Smirnov) normality test
data: x
D = 0.0319, p-value = 0.4119
> cvm.test(x)
  Cramer-von Mises normality test
data: x
W = 0.0557, p-value = 0.4308
> ad.test(x)
  Anderson-Darling normality test
data: x
A = 0.3762, p-value = 0.4103
> pearson.test(x, n.classes = 10)
  Pearson chi-square normality test
data: x
P = 7.8, p-value = 0.3506
```

Example 2—Normality Tests without 'groupingColumns'

This example uses the same data as Example 1, but this example runs the tests only on 'storeid = 1'. The syntax is slightly different.

Example SQL-MR Call (FACT + DIMENSION w/o GROUPING COLUMNS)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER (PARTITION BY 1 ORDER BY price) AS rank, * FROM
            distnmatch_normal_50_2 WHERE storeid = 1 AND price IS NOT NULL) AS
        input PARTITION BY ANY
        ON (SELECT COUNT(*) AS group_size FROM distnmatch_normal_50_2 WHERE
            storeid = 1 AND price IS NOT NULL) AS groupstats DIMENSION
        VALUECOLUMN('price')
        TESTS('KS', 'CvM', 'AD', 'CHISQ')
        DISTRIBUTIONS('NORMAL:49.97225,2.009698')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    )
    PARTITION BY 1
) ;
```

Output

Table 5 - 168: Output table

Column Name	Value
-[RECORD 1]-----+	
group_size	400
NORMAL:49.97225,2.009698_KS_statistic	0.0319503
NORMAL:49.97225,2.009698_KS_p-value	0.41181
NORMAL:49.97225,2.009698_CvM_statistic	0.0556535
NORMAL:49.97225,2.009698_CvM_p-value	0.430792
NORMAL:49.97225,2.009698_AD_statistic	0.376151
NORMAL:49.97225,2.009698_AD_p-value	0.410292
NORMAL:49.97225,2.009698_CHISQ_statistic	7.8
NORMAL:49.97225,2.009698_CHISQ_p-value	0.35056

Example 3—Test against All Continuous Distributions

This example runs all the tests on a sample generated by Beta(2,5) against all the continuous distributions.

Input

Here is a snapshot of the input data:

Table 5 - 169: Input data

times	storeid	variable_name	price
2011-11-11 11:11:11	1	variable1	0.199011
2012-12-12 12:12:12	2	variable2	0.199011
2011-11-11 11:11:11	1	variable1	0.291475
2012-12-12 12:12:12	2	variable2	0.291475
2011-11-11 11:11:11	1	variable1	0.253774
2012-12-12 12:12:12	2	variable2	0.253774
...

Example SQL-MR Call (CO-GROUP)

```

SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER
            (PARTITION BY times, storeid, variable_name ORDER BY price) AS
            rank, * FROM distnmatch_beta_2_5 WHERE price IS NOT NULL) AS
            input PARTITION BY times, storeid, variable_name
        ON (SELECT times, storeid, variable_name, COUNT(*) AS group_size FROM
            distnmatch_beta_2_5 WHERE price IS NOT NULL GROUP BY times,
            storeid, variable_name) AS groupstats PARTITION BY times, storeid,
            variable_name
        VALUECOLUMN('price')
        TESTS ('KS', 'CvM', 'AD', 'CHISQ')
        DISTRIBUTIONS('BETA:2,5', 'CAUCHY:0,2', 'CHISQ:3',
        'EXPONENTIAL:0.5', 'F:5,2', 'GAMMA:2,0.5', 'LOGNORMAL:0,0.25',
        'NORMAL:0.2892402,0.1559140', 'T:10', 'TRIANGULAR:0,0.2,1',
        'UNIFORMCONTINUOUS:0,1', 'WEIBULL:2,2')
        GROUPINGCOLUMNS('times', 'storeid', 'variable_name')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    )
    PARTITION BY times, storeid, variable_name
);

```

Example Output

This examples shows only the first record. The second record is the same except for column grouping. In addition, because the first record has close to 100 rows, only a few are shown here.

Table 5 - 170: Output table

Column Name	Value
-[RECORD 1]-----+	
times	2012-12-12 12:12:12
storeid	2
variable_name	variable2
group_size	400
BETA:2,5_KS_statistic	0.035179
BETA:2,5_KS_p-value	0.705267
BETA:2,5_CvM_statistic	0.0918871
BETA:2,5_CvM_p-value	0.145044
BETA:2,5_AD_statistic	0.692663
BETA:2,5_AD_p-value	0.564896
BETA:2,5_CHISQ_statistic	7.1
...	...
TRIANGULAR:0,0.2,1_AD_p-value	1.5e-06
TRIANGULAR:0,0.2,1_CHISQ_statistic	107.25
TRIANGULAR:0,0.2,1_CHISQ_p-value	0
UNIFORMCONTINUOUS:0,1_KS_statistic	0.395106
UNIFORMCONTINUOUS:0,1_KS_p-value	1.11022e-16
UNIFORMCONTINUOUS:0,1_CvM_statistic	25.777
...	...
WEIBULL:2,2_AD_p-value	1.5e-06
WEIBULL:2,2_CHISQ_statistic	3443.2
WEIBULL:2,2_CHISQ_p-value	0

Example 4—Test against All Discrete Distributions

This example runs all the tests on a sample generated by Poisson(5) against all the discrete distributions.

Input

Here is a snapshot of the input data:

Table 5 - 171: Input data

times	storeid	variable_name	price
2011-11-11 11:11:11	1	variable1	5
2012-12-12 12:12:12	2	variable2	5
2011-11-11 11:11:11	1	variable1	3
2012-12-12 12:12:12	2	variable2	3
2011-11-11 11:11:11	1	variable1	6
2012-12-12 12:12:12	2	variable2	6
...

Example SQL-MR Call (CO-GROUP w/ ORDER BY)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1)) OVER
            (PARTITION BY times, storeid, variable_name ORDER BY price) AS
            rank, times, storeid, variable_name, price FROM
            distnmatch_poisson_5 WHERE price IS NOT NULL GROUP BY times,
            storeid, variable_name, price) AS input PARTITION BY times,
            storeid, variable_name ORDER BY price
        ON (SELECT times, storeid, variable_name, COUNT(*) AS group_size FROM
            distnmatch_poisson_5 WHERE price IS NOT NULL GROUP BY times,
            storeid, variable_name) AS groupstats PARTITION BY times, storeid,
            variable_name
        VALUECOLUMN('price')
        TESTS('KS', 'CvM', 'AD', 'CHISQ')
        DISTRIBUTIONS('BINOMIAL:400,0.0125',
        'NEGATIVEBINOMIAL:10000,0.9995002', 'POISSON:5', 'GEOMETRIC:0.2',
        'NEGATIVEBINOMIAL:1,0.2', 'UNIFORMDISCRETE:0,12')
        GROUPINGCOLUMNS('times', 'storeid', 'variable_name')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    )
    PARTITION BY times, storeid, variable_name
);
```

Output

Only the first record (storeid = 1, variable1) is shown here. The second record is the same except for column grouping. Also, only a portion of the rows for the first record are shown.

Table 5 - 172: Output table

Column Name	Value
-[RECORD 1]-----+	
times	2011-11-11 11:11:11
storeid	1
variable_name	variable1
group_size	400
BINOMIAL:400,0.0125_KS_statistic	0.0379454
BINOMIAL:400,0.0125_KS_p-value	0.612192
BINOMIAL:400,0.0125_CvM_statistic	0.11455
BINOMIAL:400,0.0125_CvM_p-value	8.72979
BINOMIAL:400,0.0125_AD_statistic	13.1553
BINOMIAL:400,0.0125_AD_p-value	1.50002e-06
BINOMIAL:400,0.0125_CHISQ_statistic	156.8
BINOMIAL:400,0.0125_CHISQ_p-value	0
NEGATIVEBINOMIAL:10000,0.9995002_KS_statistic	0.036524
NEGATIVEBINOMIAL:10000,0.9995002_KS_p-value	0.660078
NEGATIVEBINOMIAL:10000,0.9995002_CvM_statistic	0.100235
NEGATIVEBINOMIAL:10000,0.9995002_CvM_p-value	9.97655
NEGATIVEBINOMIAL:10000,0.9995002_AD_statistic	13.0431
NEGATIVEBINOMIAL:10000,0.9995002_AD_p-value	1.50005e-06
NEGATIVEBINOMIAL:10000,0.9995002_CHISQ_statistic	156.8
NEGATIVEBINOMIAL:10000,0.9995002_CHISQ_p-value	0
POISSON:5_KS_statistic	0.0366283
POISSON:5_KS_p-value	0.656558
POISSON:5_CvM_statistic	0.100721
POISSON:5_CvM_p-value	9.92838
POISSON:5_AD_statistic	13.0659
POISSON:5_AD_p-value	1.50005e-06
...	...

Example 5—Test against All Discrete Distributions without 'groupingColumns'

This example uses the same data as example 4, but the call does not have column grouping.

Example SQL-MR Call (CO-GROUP w/ ORDER BY, w/o GROUPING COLUMNS)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1))
            OVER (PARTITION BY 1 ORDER BY price) AS rank, price FROM
            distnmatch_poisson_5 WHERE storeid = 1 AND price IS NOT NULL
            GROUP BY price) AS input PARTITION BY 1 ORDER BY price
        ON (SELECT COUNT(*) AS group_size FROM distnmatch_poisson_5 WHERE
            storeid = 1 AND price IS NOT NULL) AS groupstats PARTITION BY 1
        VALUECOLUMN('price')
        TESTS('KS', 'CvM', 'AD', 'CHISQ')
        DISTRIBUTIONS('BINOMIAL:400,0.0125',
        'NEGATIVEBINOMIAL:10000,0.9995002', 'POISSON:5', 'GEOMETRIC:0.2',
        'NEGATIVEBINOMIAL:1,0.2', 'UNIFORMDISCRETE:0,12')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    )
    PARTITION BY 1
) ;
```

Output

Table 5 - 173: Output table

Column Name	Value
-[RECORD 1]-----+	
group_size	400
BINOMIAL:400,0.0125_KS_statistic	0.0379454
BINOMIAL:400,0.0125_KS_p-value	0.612192
BINOMIAL:400,0.0125_CvM_statistic	0.11455
BINOMIAL:400,0.0125_CvM_p-value	8.72979
BINOMIAL:400,0.0125_AD_statistic	13.1553
BINOMIAL:400,0.0125_AD_p-value	1.50002e-06
BINOMIAL:400,0.0125_CHISQ_statistic	156.8
BINOMIAL:400,0.0125_CHISQ_p-value	0
NEGATIVEBINOMIAL:10000,0.9995002_KS_statistic	0.036524
NEGATIVEBINOMIAL:10000,0.9995002_KS_p-value	0.660078
NEGATIVEBINOMIAL:10000,0.9995002_CvM_statistic	0.100235
NEGATIVEBINOMIAL:10000,0.9995002_CvM_p-value	9.97655
NEGATIVEBINOMIAL:10000,0.9995002_AD_statistic	13.0431

Table 5 - 173: Output table (continued)

Column Name	Value
NEGATIVEBINOMIAL:10000,0.9995002_AD_p-value	1.50005e-06
NEGATIVEBINOMIAL:10000,0.9995002_CHISQ_statistic	156.8
NEGATIVEBINOMIAL:10000,0.9995002_CHISQ_p-value	0
POISSON:5_KS_statistic	0.0366283
POISSON:5_KS_p-value	0.656558
POISSON:5_CvM_statistic	0.100721
POISSON:5_CvM_p-value	9.92838
POISSON:5_AD_statistic	13.0659
POISSON:5_AD_p-value	1.50005e-06
POISSON:5_CHISQ_statistic	156.8
POISSON:5_CHISQ_p-value	0
...	...

distnmatch (“Best Match” Mode)

Summary

The distnmatch function finds the best matching distribution for the given data, based on the results of the “Hypothesis Test” mode of distnmatch. The following tests are available for finding the best matches:

- Anderson-Darling test
- Kolmogorov-Smirnov test
- Pearson’s Chi-squared test

The function finds a list of best matches according to each test specified.

Background

Given a one-dimensional data, you might want to know from which distribution the data might have been generated. This is useful for modeling the data and making predictions.

Usage

Syntax for real type input data (version 1.0)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER (PARTITION BY col1, col2, ..., colN
            ORDER BY column_name) AS rank, *
            FROM input_table WHERE column_name IS NOT NULL)
        AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size,
            AVG(column_name) AS mean,
            STDDEV(column_name) AS sd,
            CASE
                WHEN MIN(column_name) > 0 THEN AVG(LN(CASE WHEN column_name > 0
                    THEN column_name ELSE 1 END))
                ELSE 0
            END AS mean_of_ln,
            CASE
                WHEN MIN(column_name) > 0 THEN
                    STDDEV(LN(CASE WHEN column_name > 0 THEN
                        column_name ELSE 1 END))
                ELSE -1
            END AS sd_of_ln,
            MAX(column_name) AS maximum, MIN(column_name) AS minimum
            FROM input_table
            WHERE column_name IS NOT NULL GROUP BY col1, col2, ..., colN)
        AS groupstats DIMENSION
        VALUECOLUMN(column_name)
        [TESTS('KS', 'AD', 'CHISQ')]
        [DISTRIBUTIONS('distribution1:parameter1',...)]
        [GROUPINGCOLUMNS(col1, col2, ..., colN)]
        [MINGROUPSIZE(minGroupSize)]
        [CELLSIZE(cellSize)]
    )
    PARTITION BY col1, col2, ..., colN
    [TOP('top'))];
```

Syntax for integer type input data (version 1.0)

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts,
            SUM(COUNT(1)) OVER (PARTITION BY col1, col2, ..., colN
                ORDER BY column_name) AS rank,
            col1, col2, ..., colN, column_name
            FROM input_table
            WHERE column_name IS NOT NULL
            GROUP BY col1, col2, ..., colN, column_name)
        AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size,
            AVG(column_name) AS mean,
            STDDEV(column_name) AS sd,
            MAX(column_name) AS maximum,
            MIN(column_name) AS minimum
            FROM input_table
            WHERE column_name IS NOT NULL
```

```

        GROUP BY col1, col2, ..., colN
        AS groupstats DIMENSION
        VALUECOLUMN(column_name)
        [TESTS ('KS', 'AD', 'CHISQ')]
        [DISTRIBUTIONS ('distribution1:parameter1', ...)]
        [GROUPINGCOLUMNS (col1, col2, ..., colN)]
        [MINGROUPSIZE (minGroupSize)]
        [CELLSIZE (cellSize)]
    )
    PARTITION BY col1, col2, ..., colN
    [TOP ('top')]
) ;

```

Arguments

<i>VALUECOLUMN</i>	<i>Required</i>	Column containing the data. For continuous distributions, data type can be any numeric type. For discrete distributions, data type must be integer type.
<i>TESTS</i>	<i>Optional</i>	A list of tests the user wants to perform. Supported tests are 'KS' (Kolmogorov-Smirnov test), 'AD' (Anderson-Darling test), 'CHISQ' (Pearson's Chi-squared test). By default, all tests are run.
<i>DISTRIBUTIONS</i>	<i>Optional</i>	<p>A list of reference distributions (must be continuous distributions) from which the function will find out the best matches. Otherwise the function will find out the best matches from the most popular distributions as listed below.</p> <p>The reference distribution must be one of (continuous distributions) BETA, CAUCHY, CHISQ, EXPONENTIAL, F, GAMMA, LOGNORMAL, NORMAL, T, TRIANGULAR, UNIFORMCONTINUOUS, and WEIBULL, or (discrete distributions) BINOMIAL, GEOMETRIC, NEGATIVEBINOMIAL, POISSON, and UNIFORMDISCRETE.</p> <p>When specifying a reference distribution, you can also define parameters associated with that distribution. If you do not, estimates of the parameters for that distribution family will be used; if you do, here are the format that how you should provide parameters for each distribution:</p> <ul style="list-style-type: none"> • Continuous distributions (“Continuous distributions” on page 280) • Discrete distributions (“Discrete distributions” on page 280)
<i>GROUPINGCOLUMNS</i>	<i>Optional</i>	Column containing the group identifications over which the test will run. The function can run multiple tests for different partitions of the data in parallel. If there is no grouping column, then PARTITION BY 1 should be used. In the second ON clause, <i>col1</i> , <i>col2</i> , ..., <i>colN</i> and GROUP BY should be omitted.
<i>MINGROUPSIZE</i>	<i>Optional</i>	The minimum size of a group. Any group containing less points than this value is not considered while running the statistical tests. Default value is 50.
<i>CELLSIZE</i>	<i>Optional</i>	The number of cells that you want to discretize the continuous distribution. Default value is 10. Note that <i>MINGROUPSIZE/CELLSIZE</i> can not be less than 5, otherwise an error will be raised. Also, <i>CELLSIZE</i> must be greater than 3 if <i>NORMAL</i> distribution is tested against, and greater than 1 in all other situations.
<i>TOP</i>	<i>Optional</i>	The number of the top matching distributions you want the function to output. Default value is 1.

Continuous Distributions

Table 5 - 174: Continuous distributions

Distribution	Description
BETA: α, β	$\alpha > 0$ is the first shape parameter; $\beta > 0$ is the second shape parameter.
CAUCHY: x, θ	x , real, is the median parameter; $\theta > 0$ is the scale parameter.
CHISQ: k	k , positive integer, is the degrees of freedom.
EXPONENTIAL: θ	$\theta > 0$ is the mean parameter, which is the inverse rate.
F: d_1, d_2	$d_1 > 0, d_2 > 0$, are degrees of freedom parameters.
GAMMA: k, θ	$k > 0$ is the shape parameter; $\theta > 0$ is the scale parameter.
LOGNORMAL: μ, σ	μ , real, is the mean; $\sigma > 0$ is the standard deviation.
NORMAL: μ, σ	μ , real, is the mean; $\sigma > 0$ is the standard deviation.
T: k	k , positive integer, is the degrees of freedom.
TRIANGULAR: a, c, b	$a \leq c \leq b \ \&\& a < b$, where a is the lower limit of this distribution (inclusive), b is the upper limit of this distribution (inclusive), and c is the mode of this distribution.
UNIFORMCONTINUOUS: a, b	$a < b$, where a is the lower bound of this distribution (inclusive), and b is the upper bound of this distribution (exclusive).
WEIBULL: α, β	$\alpha > 0$ is the shape parameter; $\beta > 0$ is the scale parameter. This implementation uses the two parameter form of the distribution defined by Weibull Distribution , equations (1) and (2).

Discrete Distributions

- BINOMIAL, GEOMETRIC, NEGATIVEBINOMIAL, and POISSON distributions are on: $N=\{0,1,2,\dots\}$
- UNIFORMDISCRETE distribution is on events, using integers just as representation.

Table 5 - 175: Discrete distributions

Distribution	Description
BINOMIAL: n, p	n , positive integer, is the number of trials; p , in $[0,1]$, is the success probability in each trial.
GEOMETRIC: p	p , in $[0,1]$, is the success probability in each trial.
NEGATIVEBINOMIAL: r, p	r , positive integer, is the number of successes until the experiment is stopped; p , in $[0,1]$, is the success probability in each trial. The present implementation represents the distribution of the number of failures before r successes occur. This is the convention adopted by, for example, MathWorld , but not in Wikipedia .

Table 5 - 175: Discrete distributions (continued)

Distribution	Description
POISSON: λ	$\lambda > 0$ is the rate parameter.
UNIFORMDISCRETE:a,b	a < b, both integers, where a is the lower bound of this distribution (inclusive), and b is the upper bound of this distribution (exclusive).

Inputs

This function expects two tables.

- [Table 1](#)
- [Table 2](#)

Table 1

The first table contains the data, upon which the matching is run, in the value column. The data is first sorted (globally or within each group), and a rank is appended to each row. If you want to avoid sorting inside the database, provide the input table with a ‘rank’ column, and write use this clause instead:

```
ON SELECT * FROM input_table
```

For continuous distributions, table 1 should have the following schema:

Table 5 - 176: Table 1 schema

Column	Description
rank	For each value.
All input columns in the original order	Contain the value column as well as the grouping columns.

For discrete distributions, table 1 should have the following schema:

Table 5 - 177: Table 1 schema

Column	Description
counts	For each distinct integer.
rank	Accumulated counts order by value.
grouping columns	
value column	distinct integers

Table 2

The second table summarizes some statistics for each group. The statistics are used to estimate parameters for the distributions. If there is no grouping column, then omit the ‘GROUP BY ...’ clause.

Note the following:

- The *group_size* parameter is always required
- For the real type data, you are required to provide six aggregates:
 - mean
 - sd
 - mean_of_ln
 - sd_of_ln
 - maximum
 - minimum
- For integer type data, you are required to provide four aggregates:
 - mean
 - sd
 - maximum
 - minimum

Output

For each group (g_k), the output has the following schema:

Table 5 - 178: Output schema

Column name
grouping columns
group_size
best_match_from_KS
p-value_for_KS
best_match_from_AD
p-value_for_AD
best_match_from_CHISQ
p-value_for_CHISQ

Examples

Example 1—Match Real Type Input Data

This example shows how the function tries finding the best matching distribution for some real type input data. The data comes from several sources and are generated by different distributions.

Input

Here is a snapshot of the input table.

Table 5 - 179: Input table (distnmatch_continuous)

distribution	times	storeid	variable_name	price
normal_50_2	2011-11-11 11:11:11	1	variable1	44.6267
normal_50_2	2012-12-12 12:12:12	2	variable2	44.6267
...
beta_2_5	2011-11-11 11:11:11	1	variable1	0.575154
beta_2_5	2012-12-12 12:12:12	2	variable2	0.575154
...
t_10	2011-11-11 11:11:11	1	variable1	1.58792
t_10	2012-12-12 12:12:12	2	variable2	1.58792
...

Example SQL-MR Call

```

SELECT * FROM DistnmatchReduce (
  ON DistnmatchMultipleinput (
    ON (SELECT RANK() OVER (PARTITION BY distribution, times, storeid, variable_name
      ORDER BY price) AS rank, *
      FROM distnmatch_continuous WHERE price IS NOT NULL) AS input PARTITION BY ANY
    ON (SELECT distribution, times, storeid, variable_name,
      COUNT(*) AS group_size,
      AVG(price) AS mean,
      STDDEV(price) AS sd,
      CASE
        WHEN MIN(price) > 0 THEN AVG(LN(CASE WHEN price > 0 THEN price ELSE 1 END))
        ELSE 0
      END AS mean_of_ln,
      CASE
        WHEN MIN(price) > 0 THEN STDDEV(LN(CASE WHEN price > 0 THEN price ELSE 1 END))
        ELSE -1
      END AS sd_of_ln,
      MAX(price) AS maximum,
      MIN(price) AS minimum
      FROM distnmatch_continuous
      WHERE price IS NOT NULL
      GROUP BY distribution, times, storeid, variable_name)
    AS groupstats DIMENSION
    VALUECOLUMN('price')
    TESTS('KS', 'AD', 'CHISQ')
    GROUPINGCOLUMNS('distribution', 'times', 'storeid', 'variable_name')
    MINGROUPSIZE('50')
    CELLSIZE('10')
  )
  PARTITION BY distribution, times, storeid, variable_name
);

```

Output

Table 5 - 180: Output table (columns 1 to 6)

distribution	times	storeid	variable_name	group_size	best_match_KS
beta_2_5	2011-11-11 11:11:11	1	variable1	400	GAMMA:2.968,0.097
beta_2_5	2012-12-12 12:12:12	2	variable2	400	GAMMA:2.968,0.097
normal_50_2	2011-11-11 11:11:11	1	variable1	400	LOGNORMAL:3.91,0.04
normal_50_2	2012-12-12 12:12:12	2	variable2	400	LOGNORMAL:3.91,0.04
t_10	2011-11-11 11:11:11	1	variable1	400	T:31
t_10	2012-12-12 12:12:12	2	variable2	400	T:31

Table 5 - 181: Output table (columns 7 to 11)

p-value_KS	best_match_AD	p-value_AD	best_match_CHISQ	p-value_CHISQ
0.731083	BETA:2.157,5.300	0.890877	BETA:2.157,5.300	0.637119
0.731083	BETA:2.157,5.300	0.890877	BETA:2.157,5.300	0.637119
0.839824	GAMMA:618.48,0.08	0.868777	LOGNORMAL:3.91,0.04	0.465415
0.839824	GAMMA:618.48,0.08	0.868777	LOGNORMAL:3.91,0.04	0.465415
0.992962	T:31	0.98806	T:31	0.921005
0.992962	T:31	0.98806	T:31	0.921005

Example 2—Match Integer Type Input Data

This example shows how the function tries finding the best matching distribution for some integer type input data. The data comes from several sources and are generated by different distributions.

Input

Here is a snapshot of the input table:

Table 5 - 182: Input table (distnmatch_discrete)

distribution	times	storeid	variable_name	price
poisson_5	2011-11-11 11:11:11	1	variable1	7
poisson_5	2012-12-12 12:12:12	2	variable2	7
...
uniform_0_12	2011-11-11 11:11:11	1	variable1	6
uniform_0_12	2012-12-12 12:12:12	2	variable2	6
...

Table 5 - 182: Input table (distnmatch_discrete)

distribution	times	storeid	variable_name	price
geometric_6	2011-11-11 11:11:11	1	variable1	1
geometric_6	2012-12-12 12:12:12	2	variable2	1
...

Example SQL-MR Call

```

SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts,
            SUM(COUNT(1)) OVER (PARTITION BY distribution, times, storeid, variable_name
                ORDER BY price) AS rank,
            distribution, times, storeid, variable_name, price
        FROM distnmatch_discrete
        WHERE price IS NOT NULL
        GROUP BY distribution, times, storeid, variable_name, price)
        AS input PARTITION BY ANY
        ON (SELECT distribution, times, storeid, variable_name,
            COUNT(*) AS group_size,
            AVG(price) AS mean,
            STDDEV(price) AS sd,
            MAX(price) AS maximum,
            MIN(price) AS minimum
        FROM distnmatch_discrete
        WHERE price IS NOT NULL
        GROUP BY distribution, times, storeid, variable_name)
        AS groupstats DIMENSION
        VALUECOLUMN('price')
        TESTS('KS', 'AD', 'CHISQ')
        GROUPINGCOLUMNS('distribution', 'times', 'storeid', 'variable_name')
        MINGROUPSIZE('50')
        CELLSIZE('10')
    )
    PARTITION BY distribution, times, storeid, variable_name
);

```

Output

Table 5 - 183: Output table (columns 1 to 6)

distribution	times	storeid	variable_name	group_size	best_match_KS
geometric_6	2011-11-11 11:11:11	1	variable1	400	GEOMETRIC:0.164
geometric_6	2012-12-12 12:12:12	2	variable2	400	GEOMETRIC:0.164
poisson_5	2011-11-11 11:11:11	1	variable1	400	POISSON:5.015
poisson_5	2012-12-12 12:12:12	2	variable2	400	POISSON:5.015
uniform_0_12	2011-11-11 11:11:11	1	variable1	400	UNIFORMDISCRETE:0,12

Table 5 - 183: Output table (columns 1 to 6) (continued)

distribution	times	storeid	variable_name	group_size	best_match_KS
uniform_0_12	2012-12-12 12:12:12	2	variable2	400	UNIFORMDISCRETE: 0,12

Table 5 - 184: Output table (columns 7 to 11)

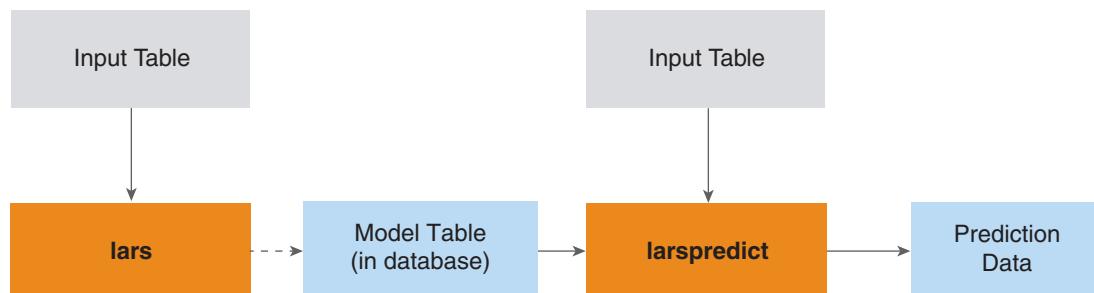
p-value_KS	best_match_AD	p-value_AD	best_match_CHISQ	p-value_CHISQ
0.944007	GEOMETRIC:0.164	1.01154e-05	BINOMIAL:100,0.5	0
0.944007	GEOMETRIC:0.164	1.01154e-05	BINOMIAL:100,0.5	0
0.709338	POISSON:5.015	1.50173e-06	BINOMIAL:100,0.5	0
0.709338	POISSON:5.015	1.50173e-06	BINOMIAL:100,0.5	0
0.582103	BINOMIAL:100,0.5	1.5e-06	UNIFORMDISCRETE:0,12	2.43839e-09
0.582103	BINOMIAL:100,0.5	1.5e-06	UNIFORMDISCRETE:0,12	2.43839e-09

LARS Functions

SUMMARY

Least Angle Regression (LARS) and its most important modification, least absolute shrinkage and selection operator (LASSO), are attractive variants of linear regression that select the most important variables, one by one, and fit the coefficients dynamically. Aster Database provides two LARS functions: lars and larspredict.

The output of the lars function is used by the larspredict function to make predictions.



Background

LARS is a model-selection algorithm, is a useful and less greedy version of traditional forward selection methods. LASSO is an attractive version of ordinary least squares (OLS) that constrains the sum of the absolute regression coefficients. LASSO is an important sparse learning method.

LASSO is a linear-regression-like method, estimating the coefficients for each input variable, which is used to make predictions for the response variable. However, compared to ordinary least squares, LASSO does the fitting in a smarter way. It always finds the most significant variables (have the greatest absolute correlation with the current residuals) in a sequential manner. In other words, it performs the *variable selection* job.

This form of fitting is very useful when you have thousands of input variables. The time complexity is the same as running linear regression, which is linear in the number of rows. In addition, LASSO can still work in some situations in which ordinary least squares cannot, such as when there is multicollinearity.

For more information about LARS, see *LEAST ANGLE REGRESSION*, Bradley Efron et al, Department of Statistics, Stanford University, 2004.

Implementation Notes

The lars function is implemented based on the 'lars' package in R. Currently, supported options are 'lar' and 'lasso'. The 'Incremental QR Decomposition' has not been implemented. There might be issues when multicollinearity appears.

lars

SUMMARY

The lars function generates a model that can be used the larspredict function to make predictions.

Usage

Syntax (version 1.0)

```
SELECT * FROM LARS (
    ON (SELECT 1)
    PARTITION BY 1
    [ DOMAIN('host_ip') ]
    [ DATABASE('db_name') ]
    [ USERID('user_id') ]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    COLUMNNAMES('Y', 'X1', 'X2', ..., 'Xp')
    [ METHOD('lar' | 'lasso') ]
    [ INTERCEPT('true' | 'false') ]
    [ NORMALIZE('true' | 'false') ]
    [ MAXITERNUM('max_iterations') ]
);
SELECT * FROM output_table_name WHERE steps > 0 ORDER BY steps;
```

This function can take at most 799 response and predictor variables.

Arguments

<i>DOMAIN</i>	Optional	IP address of the queen node. Default domain is queen of the current cluster.
<i>DATABASE</i>	Optional	The database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user. Default userid is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	A table with several columns. One of them is the response, and some other columns are predictors.
<i>OUTPUTTABLE</i>	Required	The name for the output table.
<i>COLUMNNAMES</i>	Required	Column names of the output table. Must be in this format: 'Y', 'X1', 'X2', ..., 'Xp'.
<i>METHOD</i>	Optional	Supported methods are 'lar' (least angle regression) and 'lasso'. Default is 'lasso'.
<i>INTERCEPT</i>	Optional	If 'true', an intercept is included in the model (and not penalized); otherwise no intercept is included. Default is 'true'.
<i>NORMALIZE</i>	Optional	If 'true', each variable (predictor) is standardized to have unit L2 norm; otherwise it is left alone. Default is 'true'.
<i>MAXITERNUM</i>	Optional	The default value is $8 * \min(\text{number of predictors}, \text{sample size} - \text{intercept})$.

Input

A table that contains the response column and predictor columns. All these columns must have a numeric type. Boolean is allowed only when you set both 'intercept' and 'normalize' to false.

Output

Output Table Schema

[Table 5 - 185](#) describes the schema of the output table.

Table 5 - 185: LARS Output schema

Column	Description
steps	The sequence number for each step. One lar move or lasso move represents one step.
var_id	The ID of the predictors according to the input order in the argument clause.
var_name	The column name of the predictor.
max_abs_corr	The 'modified' maximum absolute correlation (common for all active variables) between the active variables and the current residuals. It is not necessary in [-1,1].
step_length	The distance to move along the equiangular direction in each step.
intercept	The constant item in the model. It is also evolving along the path.
[Remaining columns]	The coefficient for each predictor at the end of the step.

Interpreting the Output

At the beginning of $step_i$, the variable X_k (var_id , var_name) enters into (positive var_id) / drops from (negative var_id) the regression model, and the current common correlation between active variables and current residuals is max_abs_corr .

Then, after moving along the equiangular direction for $step_length$ distance, either an inactive variable qualifies to enter into the model or a currently active variable is dropped from the model, in which the process reaches $step_{i+1}$, so the intercept and coefficients corresponding to the end of $step_i$ as well as the beginning of $step_{i+1}$.

Examples

Example 1—LAR

Example input

This example uses diabetes data used in the Efron et al “Least Angle Regression” paper. The data has 442 rows and 12 columns. The input table has this schema:

Table 5 - 186: Input table schema

id	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu	y

Where:

- id: row id
- y: response

The other columns are predictors.

This data set is a little special in that all the X values have a mean of 0 and a norm of 1. This means that:

- Whatever you specify in the 'normalize' argument does not make a difference.
- If 'intercept' is 'true', it is considered a constant along the entire path (while in general it is not necessarily the case).

Example SQL-MapReduce call

```
SELECT * FROM LARS (ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('diabetes')
    outputTable('diabetes_lars')
    columnNames('y', 'age', 'sex', 'bmi', 'map', 'tc', 'ldl', 'hdl', 'tch',
        'ltg', 'glu')
    method('lar')
    intercept('true')
    normalize('true')
    maxIterNum('20')));
SELECT * FROM diabetes_lars WHERE steps <> 0 ORDER BY steps;
```

Example output

[Table 5 - 187](#) and [Table 5 - 188](#) display the output.

Table 5 - 187: Output table (columns 1–6)

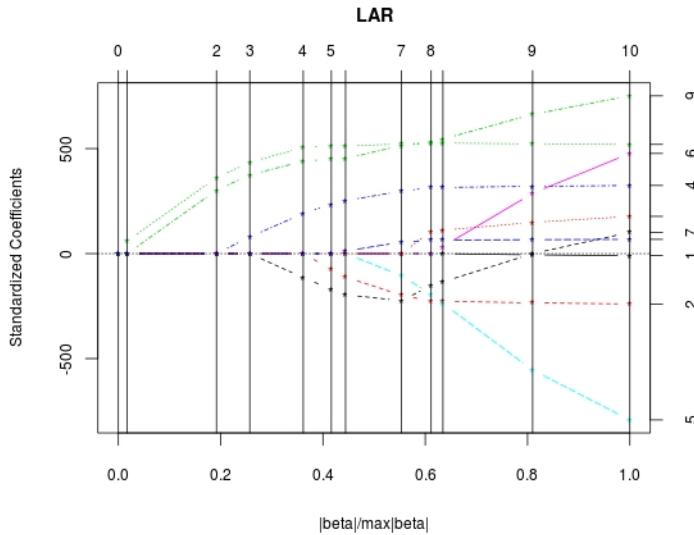
steps	var_id	var_name	max_abs_corr	step_length	intercept
1	3	bmi	949.435	60.1193	152.133
2	9	ltg	889.316	513.224	152.133
3	4	map	452.901	175.553	152.133
4	7	hdl	316.074	259.367	152.133
5	2	sex	130.131	88.6592	152.133
6	10	glu	88.7824	43.6779	152.133
7	5	tc	68.9652	135.984	152.133
8	8	tch	19.9813	54.0156	152.133
9	6	ldl	5.47747	5.56725	152.133
10	1	age	5.08918	73.5291	152.133

Table 5 - 188: Output table (columns 7–16)

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	0	60.1193	0	0	0	0	0	0	0
0	0	361.895	0	0	0	0	0	301.775	0
0	0	434.758	79.2365	0	0	0	0	374.916	0
0	0	505.66	191.27	0	0	-114.101	0	439.665	0
0	-74.9165	511.348	234.155	0	0	-169.711	0	450.667	0
0	-111.979	512.044	252.527	0	0	-196.045	0	452.393	12.0781
0	-197.757	522.265	297.16	-103.946	0	-223.926	0	514.749	54.7677
0	-226.134	526.885	314.389	-195.106	0	-152.477	106.343	529.916	64.4874
0	-227.176	526.391	314.95	-237.341	33.6284	-134.599	111.384	545.483	64.6067
-10.0122	-239.819	519.84	324.39	-792.184	476.746	101.045	177.064	751.279	67.6254

[Figure 8](#) is a representation of the results.

Figure 8: LAR results



Comparison with R

```
#### Import package and attach the data:
> library(lars)
> data(diabetes)
> attach(diabetes)

#### This shows the LARS sequence:
> object2 <- lars(x, y, type="lar", normalize=TRUE, intercept=TRUE,
+ trace=TRUE)
> object2
LAR sequence
Computing X'X' .....
LARS Step 1 : Variable 3 added
LARS Step 2 : Variable 9 added
LARS Step 3 : Variable 4 added
LARS Step 4 : Variable 7 added
LARS Step 5 : Variable 2 added
LARS Step 6 : Variable 10 added
LARS Step 7 : Variable 5 added
LARS Step 8 : Variable 8 added
LARS Step 9 : Variable 6 added
LARS Step 10 : Variable 1 added
Computing residuals, RSS etc .....

R-squared: [0.518]
Sequence of LAR moves:
      bmi  ltg  map  hdl  sex  glu  tc  tch  ldl  age
Var     3    9    4    7    2   10    5    8    6    1
Step    1    2    3    4    5    6    7    8    9   10

#### This shows the coefficients for each step:
> coef(object2)
      age      sex      bmi      map      tc      ldl      hdl      tch      ltg      glu
[1,] 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
[2,] 0.00000 60.11927 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
[3,] 0.00000 0.00000 361.89461 0.00000 0.0000 0.00000 0.0000 0.00000 301.7753 0.00000
[4,] 0.00000 0.00000 434.75796 79.23645 0.0000 0.00000 0.0000 0.00000 374.9158 0.00000
[5,] 0.00000 0.00000 505.65956 191.26988 0.0000 0.00000 -114.1010 0.0000 439.6649 0.00000
[6,] 0.00000 -74.91651 511.34807 234.15462 0.0000 0.00000 -169.7114 0.0000 450.6674 0.00000
[7,] 0.00000 -111.97855 512.04409 252.52702 0.0000 0.00000 -196.0454 0.0000 452.3927 12.07815
[8,] 0.00000 -197.75650 522.26485 297.15974 -103.9462 0.00000 -223.9260 0.0000 514.7495 54.76768
[9,] 0.00000 -226.13366 526.88547 314.38927 -195.1058 0.00000 -152.4773 106.3428 529.9160 64.48742
[10,] 0.00000 -227.17580 526.39059 314.95047 -237.3410 33.62827 -134.5994 111.3841 545.4826 64.60667
[11,] -10.01220 -239.81909 519.83979 324.39043 -792.1842 476.74584 101.0446 177.0642 751.2793 67.62539
```

```
### This shows the maximum absolute correlation for each step:
> object2$lambda
[1] 949.435260 889.315991 452.900969 316.074053 130.130851 88.782430
[7] 68.965221 19.981255 5.477473 5.089179
```

```
### This shows the step lengths:
> object2$arc.length
[1] 60.119270 513.223719 175.553223 259.367468 88.659155 43.677933
[7] 135.984080 54.015603 5.567232 73.529073
```

Example 2—LASSO

Example input

Same as example 1.

Example SQL-MapReduce call

```
SELECT * FROM LARS (
  ON (SELECT 1)
  PARTITION BY 1
  database('beehive')
  userid('beehive')
  password('beehive')
  inputTable('diabetes')
  outputTable('diabetes_lasso')
  columnNames('y', 'age', 'sex', 'bmi', 'map', 'tc', 'ldl', 'hdl', 'tch',
              'ltg', 'glu')
  method('lasso')
  intercept('true')
  normalize('true')
  maxIterNum('20')
);
SELECT * FROM diabetes_lasso WHERE steps <> 0 ORDER BY steps;
```

Example output

[Table 5 - 189](#) and [Table 5 - 190](#) display the output.

Table 5 - 189: Output table (columns 1–6)

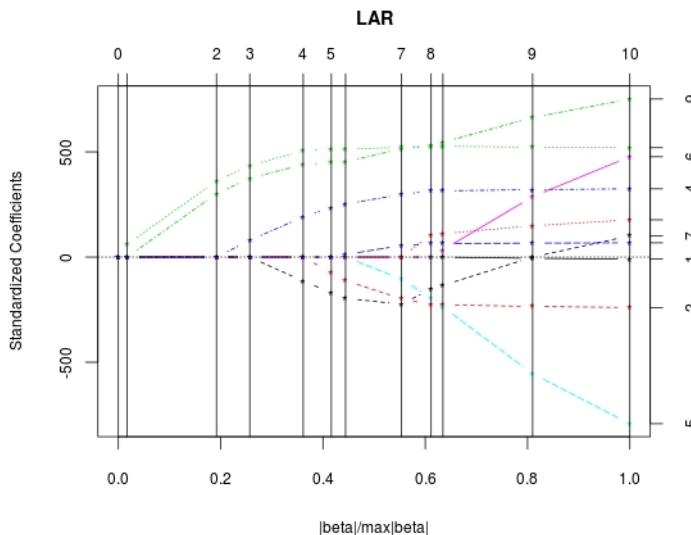
steps	var_id	var_name	max_abs_corr	step_length	intercept
1	3	bmi	949.435	60.1193	152.133
2	9	ltg	889.316	513.224	152.133
3	4	map	452.901	175.553	152.133
4	7	hdl	316.074	259.367	152.133
5	2	sex	130.131	88.6592	152.133
6	10	glu	88.7824	43.6779	152.133
7	5	tc	68.9652	135.984	152.133
8	8	tch	19.9813	54.0156	152.133
9	6	ldl	5.47747	5.56725	152.133
10	1	age	5.08918	41.9996	152.133
11	-7	hdl	2.18225	7.2707	152.133
12	7	hdl	1.31044	27.97	152.133

Table 5 - 190: Output table (columns 7–16)

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	0	60.1193	0	0	0	0	0	0	0
0	0	361.895	0	0	0	0	0	301.775	0
0	0	434.758	79.2365	0	0	0	0	374.916	0
0	0	505.66	191.27	0	0	0	-114.101	0	439.665
0	-74.9165	511.348	234.155	0	0	-169.711	0	450.667	0
0	-111.979	512.044	252.527	0	0	-196.045	0	452.393	12.0781
0	-197.757	522.265	297.16	-103.946	0	-223.926	0	514.749	54.7677
0	-226.134	526.885	314.389	-195.106	0	-152.477	106.343	529.916	64.4874
0	-227.176	526.391	314.95	-237.341	33.6284	-134.599	111.384	545.483	64.6067
-5.71894	-234.398	522.649	320.343	-554.266	286.736	0	148.9	663.033	66.331
-7.01124	-237.101	521.075	321.549	-580.439	313.862	0	139.858	674.937	67.1794
-10.0122	-239.819	519.84	324.39	-792.184	476.746	101.045	177.064	751.279	67.6254

[Figure 9](#) is a representation of the results.

Figure 9: LASSO results



Comparison with R

```
### Import package and attach the data:
> library(lars)
> data(diabetes)
> attach(diabetes)

### This shows the LARS sequence:
> object3 <- lars(x, y, type="lasso", normalize=TRUE, intercept=TRUE,
+ trace=TRUE)
> object3
LASSO sequence
Computing X'X' .....
LARS Step 1 : Variable 3 added
LARS Step 2 : Variable 9 added
LARS Step 3 : Variable 4 added
LARS Step 4 : Variable 7 added
LARS Step 5 : Variable 2 added
LARS Step 6 : Variable 10 added
LARS Step 7 : Variable 5 added
LARS Step 8 : Variable 8 added
LARS Step 9 : Variable 6 added
LARS Step 10 : Variable 1 added
Lasso Step 11 : Variable 7 dropped
LARS Step 12 : Variable 7 added
Computing residuals, RSS etc .....

R-squared: [0.518]
Sequence of LASSO moves:
  bmi ltg map hdl sex glu tc tch ldl age hdl hdl
Var    3    9    4    7    2   10   5    8    6    1   -7    7
Step   1    2    3    4    5    6    7    8    9   10   11   12
```

Statistical Analysis LARS Functions

```

#### This shows the coefficients for each step:
> coef(object3)

  age   sex   bmi   map    tc    ldl    hdl    tch    ltg    glu
[1,] 0.000000 0.00000 0.00000 0.0000 0.00000 0.0000 0.0000 0.00000 0.00000
[2,] 0.000000 0.00000 60.11927 0.00000 0.0000 0.00000 0.0000 0.0000 0.00000
[3,] 0.000000 0.00000 361.89461 0.00000 0.0000 0.00000 0.0000 0.0000 301.7753 0.00000
[4,] 0.000000 0.00000 434.75796 79.23645 0.0000 0.00000 0.0000 0.0000 374.9158 0.00000
[5,] 0.000000 0.00000 505.65956 191.26988 0.0000 0.00000 -114.1010 0.0000 439.6649 0.00000
[6,] 0.000000 -74.91651 511.34807 234.15462 0.0000 0.00000 -169.7114 0.0000 450.6674 0.00000
[7,] 0.000000 -111.97855 512.04409 252.52702 0.0000 0.00000 -196.0454 0.0000 452.3927 12.07815
[8,] 0.000000 -197.75650 522.26485 297.15974 -103.9462 0.00000 -223.9260 0.0000 514.7495 54.76768
[9,] 0.000000 -226.13366 526.88547 314.38927 -195.1058 0.00000 -152.4773 106.3428 529.9160 64.48742
[10,] 0.000000 -227.17580 526.39059 314.95047 -237.3410 33.62827 -134.5994 111.3841 545.4826 64.60667
[11,] -5.718948 -234.39762 522.64879 320.34255 -554.2663 286.73617 0.0000 148.9004 663.0333 66.33096
[12,] -7.011245 -237.10079 521.07513 321.54903 -580.4386 313.86213 0.0000 139.8579 674.9366 67.17940
[13,] -10.012198 -239.81909 519.83979 324.39043 -792.1842 476.74584 101.0446 177.0642 751.2793 67.62539

#### This shows the maximum absolute correlation for each step:
> object3$lambda
[1] 949.435260 889.315991 452.900969 316.074053 130.130851 88.782430
[7] 68.965221 19.981255 5.477473 5.089179 2.182250 1.310435

#### This shows the step lengths:
> object3$arc.length
[1] 60.119270 513.223719 175.553223 259.367468 88.659155 43.677933
[7] 135.984080 54.015603 5.567232 41.999664 7.270701
27.970023Example 3 - Compare with linear regression
The final LARS results (both 'lar' and 'lasso') are consistent with
linear regression (OLS solution). See the 'Estimate' column below and
compare with previous final LARS coefficients above (last row in each
table in blue).

#### Import package and attach the data:
> library(lars)
> data(diabetes)
> attach(diabetes)

#### Perform linear regression
> fit <- lm(y ~ x)
> summary(fit)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-155.829 -38.534 -0.227  37.806 151.355 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 152.133    2.576  59.061 < 2e-16 ***
xage        -10.012    59.749  -0.168  0.867000  
xsex       -239.819    61.222  -3.917 0.000104 ***
xbmi        519.840    66.534   7.813 4.30e-14 ***
xmap        324.390    65.422   4.958 1.02e-06 ***
xtc         -792.184   416.684  -1.901 0.057947 .  
xldl        476.746    339.035   1.406 0.160389  
xhdl        101.045    212.533   0.475 0.634721  

```

```

xtch      177.064    161.476   1.097 0.273456
xltg      751.279    171.902   4.370 1.56e-05 ***
xglu      67.625     65.984   1.025 0.305998
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 54.15 on 431 degrees of freedom
Multiple R-squared: 0.5177, Adjusted R-squared: 0.5066
F-statistic: 46.27 on 10 and 431 DF,  p-value: < 2.2e-16

```

larspredict

Summary

This is the predict function for lars. It takes in the new data and the model generated by the lars function, and outputs the predictions.

Usage

Syntax (version 1.0)

```

SELECT * FROM LARSPREDICT (
    ON input_table AS data PARTITION BY ANY
    ON model_table AS model DIMENSION
    [MODE('STEP | FRACTION | NORM | LAMBDA') ]
    [S('list_of_doubles')]
);

```

Arguments

<i>MODE</i>	Optional	The mode in which s represents. Default value is 'STEP'. Allowed modes are:
		<ul style="list-style-type: none"> • STEP: 's' indicates the steps corresponding to the steps in the model generated by the LARS function. 's' can be any real values in $[1, k]$, where k is the maximum step in the model. • FRACTION: 's' indicates the fractions of the L1 norm of the coefficients against the maximum L1 norm. The maximum L1 norm is that of the full OLS solution, which is the coefficients at the last step. 's' can be any real values in $[0, 1]$. • NORM: 's' indicates the L1 norm of the coefficients. 's' can be any real values in $[0, \text{max L1 norm}]$. For maximum L1 norm, see above. • LAMBDA: 's' indicates the maximum absolute correlations. For definition, see the description of <code>max_abs_corr</code> in Table 5 - 185. 's' can be any real values.
<i>S</i>	Required	A list of real values specifying the positions of the coefficients at which you want to generate the predictions. No duplicate values are allowed.

Input

Table 5 - 191: larspredict input tables

Alias	Description
data	A table that contains the predictor columns, similar to those in the model table.
model	The model generated by the lars function.

Output

A table contains the original data and the predictions.

Example

Input

This example uses as input the same diabetes data used in lars examples.

The model table diabetes_lars is generated by the lars function:

Table 5 - 192: diabetes_lars (columns 1–8)

steps	var_id	var_name	max_abs_corr	step_length	intercept	age	sex
0	0	norms	NaN	NaN	152.133	1	1
1	3	bmi	949.435	60.1193	152.133	0	0
2	9	ltg	889.316	513.224	152.133	0	0
3	4	map	452.901	175.553	152.133	0	0
4	7	hdl	316.074	259.367	152.133	0	0
5	2	sex	130.131	88.6592	152.133	0	-74.9165
6	10	glu	88.7824	43.6779	152.133	0	-111.979
7	5	tc	68.9652	135.984	152.133	0	-197.757
8	8	tch	19.9813	54.0156	152.133	0	-226.134
9	6	ldl	5.47747	5.56725	152.133	0	-227.176
10	1	age	5.08918	73.5291	152.133	-10.0122	-239.819

Table 5 - 193: diabetes_lars (column 1 and columns 9–16)

steps	...	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	...	1	1	1	1	1	1	1	1
1	...	60.1193	0	0	0	0	0	0	0
2	...	361.895	0	0	0	0	0	301.775	0

Table 5 - 193: diabetes_lars (column 1 and columns 9–16)

steps	...	bmi	map	tc	ldl	hdl	tch	ltg	glu
3	...	434.758	79.2365	0	0	0	0	374.916	0
4	...	505.66	191.27	0	0	-114.101	0	439.665	0
5	...	511.348	234.155	0	0	-169.711	0	450.667	0
6	...	512.044	252.527	0	0	-196.045	0	452.393	12.0781
7	...	522.265	297.16	-103.946	0	-223.926	0	514.749	54.7677
8	...	526.885	314.389	-195.106	0	-152.477	106.343	529.916	64.4874
9	...	526.391	314.95	-237.341	33.6284	-134.599	111.384	545.483	64.6067
10	...	519.84	324.39	-792.184	476.746	101.045	177.064	751.279	67.6254

Example SQL-MapReduce call

```
SELECT * FROM LARSPREDICT (
    ON diabetes AS data PARTITION BY ANY
    ON diabetes_lars AS model DIMENSION
    mode('step')
    s('1.6')
)
ORDER BY id;
```

Example output

Table 5 - 194: Output table (columns 1–7)

id	age	sex	bmi	map	tc	ldl
1	0.0380759	0.0506801	0.0616962	0.0218724	-0.0442235	-0.0348208
2	-0.00188202	-0.0446416	-0.0514741	-0.0263278	-0.00844872	-0.0191633
3	0.0852989	0.0506801	0.0444512	-0.00567061	-0.0455995	-0.0341945
4

Table 5 - 195: Output table (columns 8–13)

hdl	tch	ltg	glu	y	prediction_1.6
-0.0434008	-0.00259226	0.0199084	-0.0176461	151	154.359
0.0744116	-0.0394934	-0.0683297	-0.092204	75	150.277
-0.0323559	-0.00259226	0.00286377	-0.0259303	141	153.737
...

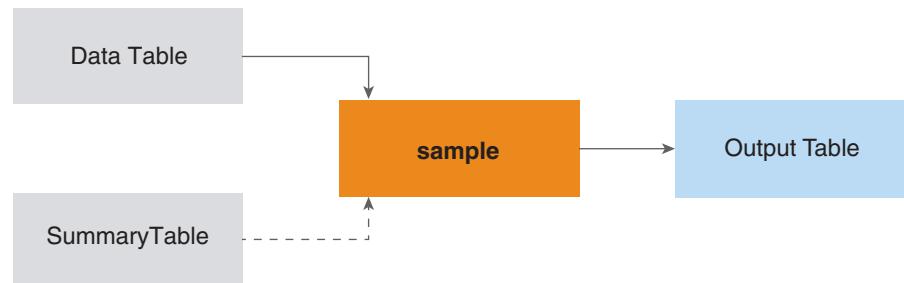
Sample

Summary

The sample function draws rows randomly from the input table. The function offers two sampling schemes:

- A simple Bernoulli (Binomial) sampling on a row-by-row basis with given sample rates
- Sampling without replacement that selects a given number of rows

The sampling can either be applied to the entire relation, which is called unconditional sampling, or be refined with conditions, which is called conditional sampling.



Note that only one random number generator is used throughout a single task for unconditional sampling, whereas a separate random number generator is created for each condition in one task for conditional sampling.

Usage

Syntax (version 1.0)

Unconditional sampling, single sample rate

```
select * from sample(
    ON ...
    SAMPLEFRACTION(f1[, f2, f3, ...])
    [Seed('seed')])
```

Unconditional sampling, total approximate sample size

```
select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    ApproximateSampleSize('size')
    [Seed('seed')])
```

conditional simple sampling, single sample rate

```
select * from sample (
    ON ...
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1'[, 'cond2', ...])
    SAMPLEFRACTION(f1[, f2, f3, ...])
    [SEED('seed')])
```

Conditional sampling, variable sample rates

```
select * from sample (
    ON ...
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1' [, 'cond2', ...])
    SAMPLEFRACTION(f1[, f2, f3, ...])
    [SEED('seed')])
```

Conditional sampling, total approximate sample size

```
select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1' [, 'cond2', ...])
    ApproximateSampleSize('total_sample_size')
    [SEED('seed')])
```

Conditional sampling, variable approximate sample sizes

```
select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1' [, 'cond2', ...])
    APPROXIMATESAMPLESIZE('s1' [, 's2', ...])
    [SEED('seed')])
```

Arguments

SAMPLEFRACTION	Required	<p>One or more sample fractions to use in the sampling of data.</p> <ul style="list-style-type: none"> • Single Sample Fraction <p>If you provide only one sample fraction, this single fraction is targeted throughout the whole population or across all the strata defined by the sample conditions.</p> <ul style="list-style-type: none"> • Variable Sample Fractions <p>If you provide multiple sample fractions, each of them is used for sampling a particular stratum defined by the condition arguments.</p> <p>To use variable sample fractions properly, the number of fractions you specify in this argument must be equal to the number of conditions in the <i>ConditionOn</i> argument.</p>
SEED	Optional	<p>A number to use for creating a random seed.</p> <p>By default the pseudo random number generator for each task uses the task ID as its random seed.</p> <p>You can customize the random seeds by passing an integer via this argument. The sample function uses the sum of this number and the task ID as the real random seed for each task.</p>

APPROXIMATESAMPLESIZE	Optional	<p>If you do not specify the SAMPLEFRACTION argument, then you must specify an approximate sample size.</p> <p>The sample function internally maps the size to sample fractions, which are then used for generating the sample data. As a result, the size of the sample is approximate, not exact.</p> <p>When you specify an approximate sample size using this argument and do not specify sample fractions using the SAMPLEFRACTIONS argument, you must provide two inputs to the sample function. You must use “data” as the alias for one of the inputs and “summary” as the alias for the other input.</p> <p>The sample function conducts the sampling on the data contained in the data input. The summary input contains the stratum count information.</p> <p>Moreover, the data input should be partitioned using the PARTITION BY ANY clause whereas the summary input should be dimensional.</p> <p>For sake of clarity, the summary input must be made up of two columns: stratum and stratum_count. The stratum column contains the conditions and stratum_count column contains the stratum sizes.</p> <p>If you are performing unconditional sampling with APPROXIMATESAMPLESIZE being specified, it is enough for the summary input to contain the stratum_count column only.</p> <p>The approximate sample size takes one or more arguments:</p> <ul style="list-style-type: none">• Total sample size<ul style="list-style-type: none">When you supply only one argument, the argument represents the total sample size for the entire population. If in addition you specify the CONDITIONON argument, the sample function proportionally generates sample units for each stratum.• Variate sample sizes<ul style="list-style-type: none">When you supply multiple sizes, each size corresponds to a stratum. The sample function generates sample units directly for each stratum based on the supplied sizes.
-----------------------	----------	--

CONDITIONONCOLUMN	Optional	<p>The sample conditions. If you use this argument, you must also use the CONDITIONON argument. Either both arguments are used, or neither is used. There are two ways to use these two arguments:</p> <ul style="list-style-type: none"> Each condition name maps to one conditional expression using a case-when statement. For example, consider an input table (Table 5 - 196) with two columns, id and score. <p>To classify the rows of the table into multiple strata, use this syntax:</p> <pre>select *, case when score <= 80 then 'fair' when score > 80 AND score < 90 then 'very good' when score >= 90 then 'excellent' end as stratum from student;</pre> <p>The resulting set appears in Table 5 - 197. Use this syntax to map the conditions to the stratum column:</p> <pre>ConditionOnColumn('stratum') ConditionOn('fair', 'excellent', 'very good');</pre> <p>If a condition name such as 'poor' is given, but it does not appear in the stratum column, the function throws an error to inform you that the condition does not exist in the summary table.</p> <ul style="list-style-type: none"> Values in a particular column can be used as sampling conditions directly. In this case, the column specified must be of a group-able type. <p>In the example shown in Table 5 - 197, to sample against only two groups of students with scores either 60 or 80, you can simply define the following without introducing a case-when statement:</p> <pre>ConditionOnColumn('score') ConditionOn('60', '80')</pre>
CONDITIONON	Optional	See description of CONDITIONONCOLUMN above.

CONDITIONONCOLUMN and CONDITIONON Examples

[Table 5 - 196](#) and [Table 5 - 197](#) are used in the description of the CONDITIONONCOLUMN and CONDITIONON arguments above.

Table 5 - 196: student table

id	score
2	60
4	73
6	78
1	80
3	95
5	88
7	98

Table 5 - 197: Result set

id	score	stratum
2	60	fair
4	73	fair
6	78	fair
1	80	fair
3	95	excellent
5	88	very good
7	98	excellent

SAMPLEFRACTION Examples

- Single Sample Fraction example:

```
select * from sample(on student SampleFraction('0.5'));
select * from sample(on student ConditionOnColumn('score')
ConditionOn('60') SampleFraction('0.5'));
```

- Variable Sample Fractions example:

```
select * from sample(
on (select *, case
when score <= 80 then 'fair'
when score > 80 AND score < 90 then 'very good'
when score >= 90 then 'excellent'
end as stratum from student)
ConditionOnColumn('stratum')
ConditionOn('fair', 'very good', 'excellent')
SampleFraction(0.3, 0.3, 0.4);
select * from sample(on (patients) ConditionOnColumn('sex')
ConditionOn('M', 'F') SampleFraction(0.02, 0.05));
```

APPROXIMATESAMPLESIZE Examples

This example draws 1200 rows from the student table.

```
select * from sample(
on (
(select stratum, count(*) as stratum_count from (select *, case
when score <= 80 then 'fair'
when score > 80 AND score < 90 then 'very good'
when score >= 90 then 'excellent'
end as stratum from student) as test group by stratum)
as summary DIMENSION
on (select *, case
when score <= 80 then 'fair'
when score > 80 AND score < 90 then 'very good'
when score >= 90 then 'excellent'
end as stratum from student)
as student_strata PARTITION BY ANY)
ConditionOnColumn('stratum')
```

```
ConditionOn('fair', 'very good', 'excellent')
ApproximateSampleSize(1200));
```

The 1st ON clause creates this dimension input named stratum_count (the dimensional information is replicated across all vWorkers):

Table 5 - 198: stratum_count

stratum	count
fair	2500
excellent	2300

The 2nd ON clause creates this fact input name student_strata:

Table 5 - 199: student_strata

id	score	stratum
1	60	fair
2	80	fair
3	80	fair
4	70	fair
6	90	excellent
5	75	fair

To simplify the summary table, you can also use a WITH clause:

```
with student_strata as (
    select *, case
        when score <= 80 then 'fair'
        when score > 80 AND score < 90 then 'very good'
        when score >= 90 then 'excellent' end as stratum
    from student)
select * from sample(
    on (select stratum, count(*) as stratum_count from
        stratified group by stratum) as summary DIMENSION
    on student_strata PARTITION BY ANY
    ConditionOnColumn('stratum')
    ConditionOn('fair', 'very good', 'excellent')
    ApproximateSampleSize(1200));
```



The summary input should summarize the population statistics faithfully. That is, the sum over stratum_count with a non-null stratum value should be equal to the total population size. If this condition does not hold, the final sample output might not approximate the target sample fractions well.



In case an expensive query is used to generate the population input, be aware that the same query might be evaluated again for generating the summary. Creating a temporary table for the input may reduce the overhead.

FMeasure

Summary

The FMeasure function calculates the accuracy of a test (usually the output of a classifier).



Background

In statistics, the F_1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score:

- p is the number of correct results divided by the number of all returned results.
- r is the number of correct results divided by the number of results that should have been returned.

The F_1 score can be interpreted as a weighted average of the precision and recall, where an F_1 score reaches its best value at 1 and its worst score at 0.

The traditional F-measure or balanced F-score (F_1 score) is the harmonic mean of precision and recall:

$$F = 2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$$

The general formula for a positive real β is:

$$F_\beta = (1 + \beta^2) * \text{precision} * \text{recall} / (\beta^2 * \text{precision} + \text{recall})$$

Usage

Syntax (version 1.0)

```
select * from FMeasure
(
    ON {table_name|view_name|(query)} PARTITION BY 1
    ExpectColumn('expect_column_name')
    PredictColumn('result_column_name')
    [Beta(beta_value)]
);
```

Arguments

EXPECTCOLUMN	Required	The name of the column with the expected category.
PREDICTCOLUMN	Required	The name of the column with the result category.
BETA	Optional	A non-negative real value. Default is 1.0.

Input

An input table, view, or query containing the test data.

Output

An output table containing the precision, recall, beta, and f-measure values.

Example

Example Input

Table 5 - 200: Input table: test_table

id	expect	predict
1	'dog'	'dog'
2	'dog'	'dog'
3	'dog'	'cat'
4	'dog'	'dog'
5	'cat'	'cat'
6	'cat'	'cat'
7	'cat'	'cat'
8	'cat'	'rat'
9	'rat'	null
10	'rat'	'rat'

Example SQL-MapReduce call

```
select * from FMeasure(
    on test_table
    PARTITION BY 1
    expectcolumn('expect')
    predictcolumn('predict')
    beta(1.0)
) ;
```

Example Output

Table 5 - 201: Output table

precision	recall	beta	fmeasure
0.777777777777	0.7	1.0	0.7368421052631577

CHAPTER 6 Text Analysis

- Levenshtein Distance
- nGram
- Text Classifier
- Text Parser (text_parser)
- Named Entity Recognition (NER)
- Sentiment Extraction Functions
- Naive Bayes Text Classifier
- TextTokenizer

Levenshtein Distance

Summary

This function computes the Levenshtein distance between two text values. The Levenshtein distance (sometimes called the “edit distance”), computes the number of edits needed to transform one string into the other, where edits include insertions, deletions, or substitutions of individual characters. This computation is useful for fuzzy matching of sequences and strings. It is one measure used to compare how “far apart” two strings are.



Background

This function is frequently used to resolve a user-entered value to a standard value, such as when a person types “Hanning Mankel” when he’s actually searching for Henning Mankell.

Usage

Permissions

You must grant EXECUTE on the function “ldist” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.1)

Use a SELECT statement to call the Levenshtein distance function:

```
SELECT *
  FROM ldist
  (
    ON {table_name | view_name | (query)}
    SOURCE (column1 [, column2,...])
    TARGET (column1)
    [THRESHOLD(value)]
    [OUTPUT_COLUMN_NAME(column_name)]
    [TARGET_COLUMN_NAME(column_name)]
    [SOURCE_COLUMN_NAME(column_name)]
    [ACCUMULATE(column1 [, column2,...])]
  );

```

Arguments

<i>SOURCE</i>	Required	A comma-delimited list of columns containing the source text whose Levenshtein distance from the target text will be calculated. SOURCE columns must be character datatype columns such as CHAR or VARCHAR.
<i>TARGET</i>	Required	Column containing the target text whose Levenshtein distance from each source text will be calculated. Must be a character datatype column.
<i>THRESHOLD</i>	Optional	Use an integer value greater than zero. If the Levenshtein distance for a (source, target) pair is equal to or less than the supplied THRESHOLD, then the Levenshtein distance is returned. If it is greater than the THRESHOLD, then -1 is returned. There is no default value; if the THRESHOLD is not set, the Levenshtein distance is returned no matter how large it is.
<i>OUTPUT_COLUMN_NAME</i>	Optional	Name you wish to apply to the output column containing the Levenshtein distance. Default is 'distance'.
<i>TARGET_COLUMN_NAME</i>	Optional	Name for the output column containing the compared target text. Default is 'target'.
<i>SOURCE_COLUMN_NAME</i>	Optional	Name for the output column containing the compared source text. Default is 'source'.
<i>ACCUMULATE</i>	Optional	List of input columns that will be passed as-is to the output.

Example

Example Input Data

Table 6 - 202: Example input table: sample_lev_input

col1	col2	col3	company	company_id
Astre	Astter	Astur	Aster	749

Example SQL-MapReduce call

```
SELECT *
  FROM ldist
  (
    ON sample_lev_input
    SOURCE ('col1', 'col2', 'col3')
    TARGET ('company')
    ACCUMULATE ('company_id')
  );
```

Example Output

Table 6 - 203: Example output table

company_id	target	source	distance
749	Aster	Astre	2
749	Aster	Astter	1
749	Aster	Astur	1

nGram

Summary

The nGram function tokenizes (or splits) an input stream of text and emits n multi-grams (called “ n -grams”) based on the specified delimiter and reset parameters. nGram provides more flexibility than standard tokenization when performing text analysis. Many two-word phrases carry important meaning (for example, “machine learning”) that unigrams (single-word tokens) do not capture. This, combined with additional analytical techniques, can be useful for performing sentiment analysis, topic identification, and document classification.



nGram considers each input row to be one document, and it returns a row for each unique n -gram in each document. Optionally, you can have nGram also return the counts of each n -gram and the total number of n -grams, per document.

Background

General background on tokenization can be found here: http://en.wikipedia.org/wiki/Lexical_analysis#Tokenizer

Usage

Permissions

You must grant EXECUTE on the function “nGram” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.2)

```
SELECT *
  FROM nGram
  (
    ON {table_name | view_name | (query)}
    TEXT_COLUMN('column_name')
    [DELIMITER('delimiter_regular_expression')]
    GRAMS(gram_number)
    [OVERLAPPING({'true' | 'false'})]
    [CASE_INSENSITIVE({'true' | 'false'})]
    [PUNCTUATION('punctuation_regular_expression')]
    RESET('reset_regular_expression')
    [TOTAL]
    [ACCUMULATE('column_name [, ...]')]
    [NGRAM_COLUMN_NAME('column_name')]
    [COUNT_COLUMN_NAME('column_name')]
  );
;
```

Arguments

<i>TEXT_COLUMN</i>	Required	Name of the column whose contents will be tokenized; only one column is permitted
<i>DELIMITER</i>	Optional	In the input text, the DELIMITER is the character or string that divides one word from the next. You can use a regular expression as the DELIMITER value. This is useful, for example, if you wish to recognize both tabs and space characters as delimiters. Default is a single space character (' ').
<i>GRAMS</i>	Required	Integer specifying the desired length, in words, of each <i>n</i> -gram (that is, the value of <i>n</i>).
<i>OVERLAPPING</i>	Optional	A true or false value that determines if you allow for overlapping <i>n</i> -grams. When running in overlapping mode, each word in each sentence is the start of an <i>n</i> -gram, provided enough words follow it (in the same sentence) to form a whole <i>n</i> -gram of the size you've specified. See <i>RESET</i> for information on sentences. Default is 'true', which allows overlapping.
<i>CASE_INSENSITIVE</i>	Optional	A true or false value that specifies whether the function will leave the lettercase of the text as-is, or convert all letters to lowercase. Default is 'true', which converts all text to lowercase.
<i>PUNCTUATION</i>	Optional	A regular expression that specifies the punctuation characters nGram will remove before it evaluates the input text. These characters are removed and not replaced with any character, so that, for example, “hocus-pocus” becomes “hucuspocus”. The default set of PUNCTUATION characters that are removed includes `~#^&*() -

<i>RESET</i>	Optional	A regular expression listing one or more punctuation characters or strings, any of which the nGram function will recognize as the end of a sentence of text. The end of each sentence resets the search for n -grams, meaning that nGram discards any partial n -grams and proceeds to the next sentence to search for the next n -gram. In other words, no n -gram can span two sentences. The default set of RESET characters includes . , ? !
<i>TOTAL</i>	Optional	A true or false value that specifies whether nGram will return a total n -gram count for the document. Each row is considered to be one document. (Note that this count is not the distinct number of n -grams, but rather the total number of n -grams in the document.) Default is 'false'. If set to 'true' then the column returned is named <i>totalcnt</i> .
<i>ACCUMULATE</i>	Optional	A list of columns you want to return for each word; note that the columns accumulated cannot have the same names as those specified by <i>NGRAM_COLUMN_NAME</i> and <i>COUNT_COLUMN_NAME</i> . By default all input columns are emitted.
<i>NGRAM_COLUMN_NAME</i>	Optional	This is the name of the column for the n -grams generated. Default is 'ngram'.
<i>COUNT_COLUMN_NAME</i>	Optional	This is the name of the count column. Default is 'frequency'.

Output

The output columns include columns specified in the *ACCUMULATE* clause and the column containing the n -gram and the count of occurrences of that n -gram.

Example

Example Input Data

Table 6 - 204: Example input table: my_docs

id	src	txt
1	wikipedia	the Quick brown fox jumps over the lazy dog
2	sampleddoc	hello world. again, I say hello world

Example SQL-MapReduce call

```
SELECT * FROM nGram (ON my_docs
    TEXT_COLUMN('txt')
    DELIMITER(' ')
    GRAMS(2)
    OVERLAPPING('true')
    CASE_INSENSITIVE('true')
    PUNCTUATION('\'[.,?!\']')
    RESET('\'[.,?!\']')
    ACCUMULATE('id','src'))
ORDER BY id;
```

Example output

Table 6 - 205: Example output from nGram

id	src	ngram	frequency
1	wikipedia	the quick	1
1	wikipedia	quick brown	1
1	wikipedia	brown fox	1
1	wikipedia	fox jumps	1
1	wikipedia	jumps over	1
1	wikipedia	over the	1
1	wikipedia	the lazy	1
1	wikipedia	lazy dog	1
2	sampledoc	hello world	2
2	sampledoc	i say	1
2	sampledoc	say hello	1

The output includes all the columns specified in the OUTPUT clause, plus an *ngram* column and a count (“cnt”) column.

Error Messages

You may encounter the following error when you run the nGram function:

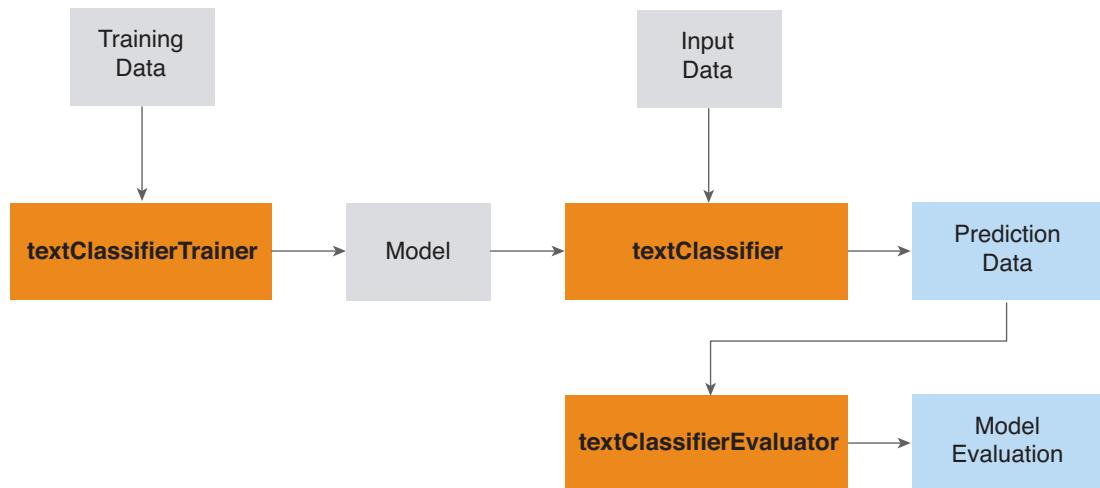
ERROR: SQL-MR function NGRAM failed: GRAMS argument should be positive.

REASON: The GRAMS argument must be positive.

Text Classifier

Summary

Text classification is the task of choosing the correct class label for a given text input. In basic text classification tasks, each input is considered in isolation from all other inputs, and the set of class labels is defined in advance.



Usage

Text Classifier includes three SQL-MR functions as follows:

- [TextClassifierTrainer](#): a training function to train the text classifier and create a model
- [TextClassifier](#): a prediction function to perform the text classification
- [TextClassifierEvaluator](#): an evaluator function to evaluate the trained classifier model.

Permissions

Before running the TextClassifierTrainer, TextClassifier and TextClassifierEvaluator functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

TextClassifierTrainer

Summary

TextClassifierTrainer is a reduce function that trains a machine learning classifier for text classification. The training result is saved as a file. Then you must be install in Aster Database before you can use it with [TextClassifier](#).

Background

The text classification process consists of two stages, like most common classification processes. Stage one is training the model, and stage two is predicting or classifying.

For the training process, these are the steps:

- 1 Use natural language processing (NLP) functionality such as tokenization, stemming, and stop words to perform preprocessing of the text data and provide tokens.
- 2 Use statistical measures to select a subset from all the tokens that preprocessing produced, (known as term filtering), then generate the feature for each word in the subset.
- 3 Use machine learning algorithms to train a classifier.

For the predicting process, the prediction function uses the trained classifier to determine the class label or category of the input text or document.

Usage

Syntax (version 1.1)

```
SELECT *
FROM TextClassifierTrainer(
    ON (SELECT 1) PARTITION BY 1
    INPUTTABLE('inputTableName')
    TEXTCOLUMN('textColumnName')
    CATEGORYCOLUMN('catColumnName')
    MODELFILE('modelFileName')
    CLASSIFIERTYPE('classifierType')
    [CLASSIFIERPARAMETERS('name:value')]
    [NLPPARAMETERS('name:value'[, ...])]
    [FEATURESELECTION('DF: [min:max] ')]
    [DATABASE('beehive')]
    [USERID('dbUserId')]
    PASSWORD('dbUserPassword')
    [DOMAIN('ip')])
);
```

Arguments

<i>INPUTTABLE</i>	Required	The table that holds the documents to use for training the model.
<i>TEXTCOLUMN</i>	Required	The name of the column whose content will be scanned. Only one column is permitted.
<i>CATEGORYCOLUMN</i>	Required	The name of the column that contains the category of the training documents.
<i>MODELFILE</i>	Required	The name of the model file to be generated. The name specified must conform to these rules: <ul style="list-style-type: none"> • can only contain the characters a-z, A-Z, 0-9, period (.), and underscore (_). • length must be greater than 0 • maximum length is 32 • can't start or end with '.'
<i>CLASSIFIERTYPE</i>	Required	Either MaxEnt or KNN classifier type is supported.

<i>CLASSIFIERPARAMETERS</i>	Optional	<p>Used to pass in the parameters for the specified classifier. This parameter is specified as name:value pairs. The parameter name and value may be different, depending on the specified <i>CLASSIFIERTYPE</i>.</p> <p>This version supports a single parameter, which is 'compress'. This parameter is used for the <i>CLASSIFIERTYPE</i> KNN. The value of compress must be greater than 0 and less than 1. For example, assume that the number of training documents is 100. Specifying <code>ClassifierParameters ('compress:0.6')</code> means that the 100 training documents will be clustered into 60 groups, and the model will use the center of each group as the feature vector.</p>
-----------------------------	----------	--

- NLPPARAMETERS** Optional Used to pass in the parameters for pre-processing. Supported values are:
- 'tokenDictFile': To segment some phrases to tokens, you can put them in a file with the format of one phrase per line, and install the file to Aster Database using the \install command in ACT. Then, you can pass in the file name using this parameter.
 - 'stopwordsFile': To filter some stop words out before tokenization occurs, you can specify them in a file with the format of one word per line, and install the file to Aster Database using the \install command in ACT. Then, you can pass in the file name using this parameter. Here is an example of a stop word file:

stop_word_file.txt	
1	a
2	an
3	the
4	and
5	this
6	with
7	they
8	but
9	will

- 'useStem': [true|false] If 'true' is specified, tells the function to do stemming after tokenization. Default is 'false'.
- 'stemIgnoreFile': To omit some words ('news', for instance) from being stemmed, you can put them in a file with the format of one word per line. Then install the file to Aster Database using the \install command in ACT, and pass in the file name using this parameter. Note that if useStem is 'false', providing this parameter will cause an exception.
- 'useBgram' (optional): [true|false] If 'true', tells the function to use bigram when analyzing tokens. Bigram takes into account the proximity of tokens to one another when analyzing adjacent tokens. Default is 'false'.
- 'language': Tells the function the language of the input text. Currently supports en, zh_CN (Simple Chinese), and zh_TW (Traditional Chinese). The default value is en. For zh_CN and zh_TW, the function will ignore the parameters useStem and stemIgnoreFile.

Each of these parameters is optional. This example specifies several pre-processing parameters:

```
NlpParamaters('tokenDictFile:token_dict.txt','stopwordsFile:fileName',
              'useStem:true','stemIgnoreFile:fileName','useBgram:true',
              'language:zh_CN')
```

FEATURESELECTION Optional Specifies the feature selection method to use. This version of the function supports DF selector (a selector that uses document frequency):

DF: The value specified must be greater than 0 but not greater than 1. For example, specifying FeatureSelection('DF: [0.1, 0.9]') means that only the tokens appearing in more than or equal to 10 percent of the documents and less than or equal to 90 percent of the documents will be selected as feature terms.

Note that DF[:maxValue] or DF[:minValue:] are also acceptable. If the value is [maxValue:minValue], the function will use the bigger value as maxValue and the smaller one as minValue.

Input Data

The input data is a table containing a column with the documents to be used to train the model.

Output

The trained model with the name specified by modelFile will be installed in the database using the \install command in ACT.

Example for TextClassifierTrainer

Example Input Data

The input data is a table of documents called train_docs.

You can create this table and insert the sample data using the following SQL:

```
CREATE FACT TABLE train_docs(id int, category varchar(10), content
varchar(15000)) DISTRIBUTE BY HASH(id);

INSERT into train_docs VALUES(1, 'finance', 'The World Bank estimates
that the economics will still fluctuate for a long time.');

INSERT into train_docs VALUES(2, 'sport', 'China won the 28th football
World Cup.');
```

Table 6 - 206: Example input table: train_docs

id	category	content
1	finance	The World Bank estimates that the economics will still fluctuate for a long time.
2	sport	China won the 28th football World Cup.

Example SQL-MapReduce call

```
SELECT *
FROM TextClassifierTrainer(
    ON (SELECT 1) PARTITION BY 1
    PASSWORD('beehive')
    InputTable('train_docs')
    TextColumn('content')
    CategoryColumn('category')
    ModelFile('knn.bin')
    ClassifierType('knn')
    ClassifierParameters('compress:0.8')
    NLPPParameters('useStem:true')
    FeatureSelection('DF: [0.1:0.9] ')
)
;
```

Example Output

The output of TextClassifierTrainer consists of the model itself, and a table with the results from generating the model.

Table 6 - 207: Example output table

train_result

Model generated.
 Training time(s): 0.065
 File name: knn.bin
 File size(KB): 0
 Model successfully installed

TextClassifier

Summary

A map function that uses the specified model to predict the category for input text.

Syntax (version 1.1)

```
SELECT *
FROM TextClassifier(
    ON inputTable
    TEXTCOLUMN('columnName')
    Model('modelName')
    [ACCUMULATE('columnName1', 'columnName2', ..., 'columnNameN')]
)
;
```

Arguments

<i>TEXTCOLUMN</i>	Required	Specifies the column of the input table that contains the text to be used for predicting classification.
<i>Model</i>	Required	The model that will be used to perform prediction. The model should be installed previously using the \install command in ACT.
<i>ACCUMULATE</i>	<i>Optional</i>	The columns of the inputTable and will be emitted as a part of the output.

Input Data

The input table must have a column with the text to be used for prediction.

Output

<i>out_category</i>	The category to which the text is predicted to belong. If the text content is empty or doesn't contain any of the feature items in the model, NULL will be output.
---------------------	--

Example for TextClassifier

Example Input Data

The input data is a table of documents called test_docs.

You can create this table and insert the sample data using the following SQL:

```
CREATE FACT TABLE test_docs(id int, content varchar(1024), category
varchar(10)) DISTRIBUTIVE BY HASH(id);

INSERT INTO test_docs VALUES(1, 'The economy is not as good as it was
last year.');

INSERT INTO test_docs VALUES(2, 'football is popular.');
```

Table 6 - 208: Example input table: test_docs

id	content
1	The economy is not as good as it was last year.
2	football is popular.

Example SQL-MapReduce call

After training the a model named knn.bin, we can use it to do prediction, using a SQL-MR call such as:

```
SELECT *
FROM TextClassifier(
    ON test_docs
    TextColumn('content')
    Model('knn.bin')
);
```

Example Output

Table 6 - 209: Example output table

out_category
sport
finance

TextClassifierEvaluator

Summary

This partition function is used to evaluate the precision, recall and f-measure of a new trained model.

Syntax (version 1.1)

```
SELECT * FROM TextClassifierEvaluator
(
    ON {table_name|view_name|(query)}
    EXPECTCOLUMN('expect_column_name')
    PREDICTCOLUMN('result_column_name')
    PARTITION BY 1
);
```

Arguments

<i>EXPECTCOLUMN</i>	Required	The name of the column with the expected category. This is the category known to be the correct classification.
<i>PREDICTCOLUMN</i>	Required	The name of the column with the result category. This is the category assigned by the prediction function TextClassifier.

Input Data

A table with a column containing the predicted classification and a column containing the known correct classification.

Output

Evaluation result, including precision, recall and f-measure of predictions.

Example for TextClassifierEvaluator

Example Input Data

The input data are the classification output from TextClassifier and the corresponding known correct classifications to use in comparing the results for accuracy.

```
update test_docs set category ='sport' where id = 2;
update test_docs set category ='finance' where id = 1;
```

Table 6 - 210: Example input table: test_docs

id	content	category
1	The economy is not as good as it was last year.	finance
2	football is popular.	sport

Example SQL-MapReduce call

```
SELECT * FROM TextClassifierEvaluator(
    ON TextClassifier(
        ON (SELECT * FROM test_docs)
        TextColumn('content')
        Accumulate('category')
        Model('knn.bin')
    )
    PARTITION BY 1
    ExpectColumn('category')
    PredictColumn('out_category')
) ;
```

Example Output

Table 6 - 211: Example output table

precision	recall	f-measure
1	1	1

Error Messages

The function does not report any specific error, except the invalid parameters error when unexpected parameters are specified.

Text Parser (text_parser)

Summary

Text parser (“text_parser” formerly “tokenize_cnt”) is a general tool for working with text fields. It can tokenize an input stream of words, optionally stem them, then emit these words in one row, or emit each individual word in a row, with an optional count of the number of times it appears.

Text parser is widely used to process text data.

For English language text, there are several key parts to take into account when parsing text:

- Punctuating sentences
- Delimiting a sentence into words
- Filtering stop words
- Stemming words

When invoked, the function reads the full document into a memory buffer and creates a hash table. The dictionary for a document should not exceed the available memory on the machine. This assumption is reasonable, since, a million-word dictionary with an average word length of ten bytes requires only 10 MB of memory.

The text_parser function uses Porter2 as the stemming algorithm.



Background

General background on tokenization can be found here: http://en.wikipedia.org/wiki/Lexical_analysis#Tokenizer

For more information about stemming, refer to: <http://en.wikipedia.org/wiki/Stemming>

Usage

Permissions

You must grant EXECUTE on the function “text_parser” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.1)

```

SELECT *
  FROM text_parser
  (
    ON {table_name|view_name|(query)}
    [PARTITION BY expression [, ...]]
    TEXT_COLUMN('text_column_name')
    [CASE_INSENSITIVE('true' | 'false')]
    [STEMMING('true' | 'false')]
    [DELIMITER('delimiter_regular_expression')]
    [TOTAL('true' | 'false')]
    [PUNCTUATION('punctuation_regular_expression')]
    [ACCUMULATE('column [, ...]')]
    [TOKEN_COLUMN_NAME('token_column_name')]
    [FREQUENCY_COLUMN_NAME('frequency_column_name')]
    [TOTAL_COLUMN_NAME('total_column_name')]
    [REMOVE_STOP_WORDS('true' | 'false')]
    [POSITION_COLUMN_NAME('position_column_name')]
    [LIST_POSITIONS('true' | 'false')]
    [OUTPUT_BY_WORD('true' | 'false')]
    [STEMMING_EXCEPTIONS('exception_rule_file')]
    [STOP_WORDS('stop_word_file')]
  );

```

Arguments

<i>TEXT_COLUMN</i>	Required	Name of the column whose contents will be tokenized. Only one column is permitted.
<i>CASE_INSENSITIVE</i>	Optional	[true false] Treat text as-is (false) or convert to all lowercase (true); Default is 'true'. Note that if the STEMMING argument is set to 'true', tokens will always be converted to lowercase, so this option will be ignored.
<i>STEMMING</i>	Optional	[true false] If true, apply Porter2 Stemming to each token to reduce it to its root form. Default is 'false'.
<i>DELIMITER</i>	Optional	Regular expression of character or string used to split words. Default value is '[\t\b\f\r]+').
<i>TOTAL</i>	Optional	[true false] If 'true' returns a column showing the total number of words in the document. Default is 'false'.
<i>PUNCTUATION</i>	Optional	List of punctuation characters to be removed, written as a regular expression. To make the stemming work better, a suggested list of punctuation to specify for this argument is '\[\.\,\?\!\:\;~()\]\]+'. Default is [.,!?].
<i>ACCUMULATE</i>	Optional	List of columns you want to return in addition to the parse word; note that no output column name can be the same as the TOKEN_COLUMN_NAME or TOTAL_COLUMN_NAME. By default, if ACCUMULATE is not selected, all input columns are returned.
<i>TOKEN_COLUMN_NAME</i>	Optional	Name of the column containing tokens. Default is 'token').
<i>FREQUENCY_COLUMN_NAME</i>	Optional	Name of the column containing frequency counts. Default is 'frequency'.
<i>TOTAL_COLUMN_NAME</i>	Optional	Name of the column containing the total count for the document. Default is 'total_count'.
<i>REMOVE_STOP_WORDS</i>	Optional	[true false] If true, ignore certain common words when parsing the text. Default is 'false'.

<i>POSITION_COLUMN_NAME</i>	Optional	Name of the column containing the position of a word within a document. Default is 'position'.
<i>LIST_POSITIONS</i>	Optional	[true false] Return position of a word in list form (if 'true'), or emit a different row for each occurrence (if 'false'). Default is 'false'.
<i>PARTITION BY</i>	Optional	The function can be invoked as either a row function or a partition function. If a <i>partition by</i> clause is specified, it will assume that all rows in any given partition constitute a single document. If no <i>partition by</i> clause is specified, it is invoked as a row function, and the function assumes each individual row constitutes a single document.
<i>OUTPUT_BY_WORD</i>	Optional	[true false]. If 'true', the output remains the same as the original function (from a sentence to a column of unique tokens). If 'false', the output will still be a sentence as the input, and further each word in the sentence will be stemmed if 'STEMMING' argument is true. The arguments 'LIST_POSITIONS' and 'FREQUENCY_COLUMN_NAME' will be invalid if 'OUTPUT_BY_WORD' is set to 'false'. Default is 'true'.
<i>STEMMING_EXCEPTIONS</i>	Optional	The location of the file that contains the list of exception rules with which the stemmer should comply. An exception rule consist of two strings; the former is the exception word, and the latter is the expected-to-be-stemmed form of this exception word. The delimiter between the two strings must be white spaces. Each exception rule is specified in a separate line. Here is an example of an exception word file:

```
exception_rule_file.txt
1 bias bias
2 news news
3 goods goods
4 lying lie
5 dying die
6 ugly uglier
7 sky sky
8 early earli
```

<i>STOP_WORDS</i>	Optional	The location of the file that contains stop words that should be ignored when parsing text. Each stop word is specified in a separate line. Here is an example of a stop word file:
-------------------	----------	--

```
stop_word_file.txt
1 a
2 an
3 the
4 and
5 this
6 with
7 they
8 but
9 will
```

As for the reasons why considering 'OUTPUT_BY_WORD', 'STEMMING_EXCEPTIONS' and 'STOP_WORDS' as arguments, the "Update to Text Parser Documentation" may be helpful. It also gives examples of stemming exception file and stop word file.

Output

If *OUTPUT_BY_WORD* is set to 'true', the output will be a column (named *TOKEN_COLUMN_NAME*) for each unique token that is found in the specified *TEXT_COLUMN*, and a column (named *FREQUENCY_COLUMN_NAME*) with a count of its occurrence in the input.

If *OUTPUT_BY_WORD* is 'false', a row in the column specified by *TEXT_COLUMN* will still be outputted as a row in the column specified by *TOKEN_COLUMN_NAME*, and the tokens in the output will be in the same order as the input.

The output also contains all of the columns specified in *ACCUMULATE* clause.

Example

Example Input Data

Table 6 - 212: Example input table: my_docs

id	src	txt
1	wikipedia	the Quick brown fox jumps over the lazy dog
2	sampledoc	hello world. again, i say hello world

Example 1 SQL-MapReduce call

```
SELECT * FROM text_parser(
  ON my_docs
  TEXT_COLUMN('txt')
  CASE_INSENSITIVE('true')
  STEMMING('true')
  PUNCTUATION('[\u00a0.,?!\u00b7;\u00e7()\\]+')
  LIST_POSITIONS('true')
  ACCUMULATE('id','src')
) ;
```

In this example, note that since *OUTPUT_BY_WORD* is not specified, the default value of 'true' is used.

Example 1 Output

Table 6 - 213: Example output table

id	src	token	frequency	position
1	wikipedia	the	2	0,6
1	wikipedia	quick	1	1
1	wikipedia	brown	1	2
1	wikipedia	fox	1	3
1	wikipedia	jumps	1	4
1	wikipedia	over	1	5
1	wikipedia	lazy	1	7
1	wikipedia	dog	1	8
2	sampledoc	hello	2	0,5
2	sampledoc	world	2	1,6
2	sampledoc	again	1	2
2	sampledoc	i	1	3
2	sampledoc	say	1	4

Example 2 SQL-MapReduce call

```
SELECT * FROM text_parser( ON my_docs
  TEXT_COLUMN('txt')
  CASE_INSENSITIVE('true')
  stemming('true')
  OUTPUT_BY_WORD('false')
  PUNCTUATION('[\\"\\[.,?\\!:\\;~()\\\\]]+')
  ACCUMULATE('id','src')
) ;
```

In this example, note that since *OUTPUT_BY_WORD* is set to 'false'.

Example 2 Output

Table 6 - 214: Example output table

id	src	tokens
1	wikipedia	the quick brown fox jump over the lazy dog
2	sampledoc	hello world again i say hello world

Error Messages

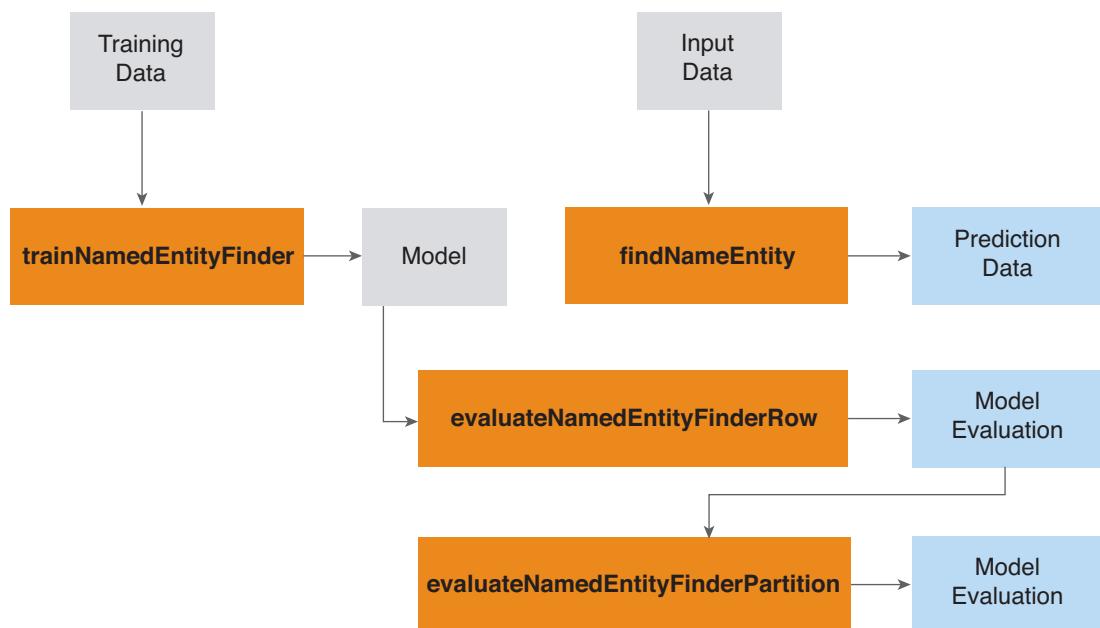
You will receive error messages under the following conditions:

- ERROR: If input parameters are not properly specified (wrong data type or values outside allowed range).
- ERROR: If the data type of the text_column parameter is not of type character varying.
- ERROR: If needed columns are missing from the relation named in the ON clause.
- ERROR: If any columns in your ACCUMULATE clause have the disallowed column name "token" or "frequency".
- ERROR: The argument 'STEMMING_EXCEPTIONS' is specified however the file cannot be located.
- ERROR: The argument 'STOP_WORDS' is specified however the stop word file cannot be located.

Named Entity Recognition (NER)

Summary

Named entity recognition (NER) is a process of finding instances of specified entities in text. For example, a simple news named-entity recognizer for the English language might find the person mentioned (John J. Smith) and the location mentioned (Seattle) in the text “John J. Smith lives in Seattle”.



Background

SQL is not a convenient way to do this type of searching. For each type of item you want to find, you would need to issue an SQL query, or merge together multiple SQL subqueries that are joined by OR operators. In addition, you would need a mechanism to label extracted fields that were found.

Usage

We use three SQL-MR functions to extract the multiple entities in text content, train and evaluate the data models.

- [FindNamedEntity](#): Extracts all the specified name entities from the input document by statistical models, regular expressions or a dictionary.
- [TrainNamedEntityFinder](#): Trains statistical models with labeled data. The input labeled data has an XML format.
- [EvaluateNamedEntityFinderRow](#) and [EvaluateNamedEntityFinderPartition](#): Evaluates the statistical models with labeled data. The format for the input labeled data is the same as is used for training.

The functions support the following embedded entity types: "person", "location", "organization", "phone", "date", "time", "email" and "money". The types "person", "location", and "organization" will use the embedded maximum entropy data model, and the types "date", "time", "email" and "money" will use embedded regular expressions. If the user specifies these entity names, the default model types and model file names will be invoked. The user could extract "all" the entities using one function call.

Permissions

Before running the FindNamedEntity, TrainNamedEntityFinder, EvaluateEntityFinderRow, and EvaluateEntityFinderPartition functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

FindNamedEntity

The FindNamedEntity function evaluates the input text, identifies tokens based on the specified model, and outputs the tokens with detailed information. The function does not identify sentences; it simply tokenizes. Token identification is not case sensitive.

Syntax (version 1.0)

```
SELECT * FROM FindNamedEntity(
    ON {table_name|view_name|(query)} PARTITION BY ANY
    ON (configure_table) as ConfigureTable DIMENSION

    TEXT_COLUMN('text_column_name')
    MODEL('entity_type_name' [ ':' 'model_type' ':'
        'model_file_name|regular_expression'] [, ...])
    [SHOW_CONTEXT('position_number')]
    [ENTITY_COLUMN ('column_name')]
    [OUTPUT_COLUMN ('column_name [, ...]')]);
```

Note: When using `FindNamedEntity` with multiple inputs, the following requirements apply:

- If the input is a query, you must map it to an alias.
- The `configure_table` input must have the alias name "ConfigureTable".

Arguments

<code>TEXT_COLUMN</code>	Required	Name of the column whose contents will be scanned. Only one column is permitted.
<code>MODEL</code>	Required	<p>List of model items. Each item is a triple, of which:</p> <ol style="list-style-type: none"> 1 The first parameter is the entity type name (for example, PERSON, LOCATION, and EMAIL) and will appear in the output. 2 The second parameter is the model type, there are four possible types: <ul style="list-style-type: none"> • max entropy—means it is a maximum entropy language model generated by training; • rule—means it is a rule based model, which is a plain text file: one regular expression per line; • dictionary—means it is dictionary based model, which is a plain text file: one word per line; • reg exp—means the entities will be extracted by the following regular expression. Not like "rule" which use pre-stored regular expressions, "reg exp" use regular expressions in the SQL-MR statement. 3 The third parameter is the model file name (for "max entropy/rule/dictionary" types) or the regular expression (for "reg exp" type). <p>For example, consider <code>MODEL('location:max entropy:en-ner-location.bin')</code>. In this model, the entity type is location, the model type is "max entropy", and the model file is "en-ner-location.bin".</p> <p>If there is no input <code>configure_table</code>, the <code>MODEL</code> argument is required parameter; otherwise, it is optional. The default value is 'all', which means that all the models in the <code>configure_table</code> are loaded.</p> <p>You can use shortcuts for the models in the <code>configure_table</code>. For example, if the <code>configure_table</code> has this row:</p> <p>(organization, max entropy, en-ner-organization.bin)</p> <p>You can use <code>MODEL('organization')</code> as a shortcut for <code>MODEL('organization:max entropy:en-ner-organization.bin')</code>.</p> <p>Note: The maximum entropy type model consumes a considerable amount of memory. For the default "all" model, the JVM of the worker node should have more than 2GB of memory.</p>
<code>SHOW_CONTEXT</code>	Optional	Whether to output the position information for each name entity. If you set this argument to a position number n , the preceding n words, the entity itself, and the following n words are omitted. For example, if entity is \$500 and $n=3$, the context column is "entity has gained \$500 thousand in its equity." If this argument is absent, no information is generated.
<code>ENTITY_COLUMN</code>	Optional	Name of the column containing entity names. Default is 'entity'.
<code>OUTPUT_COLUMN</code>	Optional	List of columns to return in the output table. The output column name cannot be the same as the <code>ENTITY_COLUMN</code> . If this argument is absent, this function includes all input columns in the output.

Creating the Table of Default Models

Before using FindNamedEntity, you must create the table of default models. Use this command to create the table:

```
create dimension table nameFind_configure(model_entitytype varchar,
model_method varchar, model_file varchar);
```

The default models are provided with the SQL-MR functions. All of them are English language models. Before using the models, install them using the \install command in ACT, and create a default configure_table.

```
drop table if exists nameFind_configure;
create dimension table nameFind_configure
    (model_name varchar, model_type varchar, model_file varchar);
insert into nameFind_configure values('person','max entropy','en-ner-person.bin');
insert into nameFind_configure values('location','max entropy','en-ner-location.bin');
insert into nameFind_configure values('organization','max entropy','en-ner-
organization.bin');
insert into nameFind_configure values('date','rules','date.rules');
insert into nameFind_configure values('time','rules','time.rules');
insert into nameFind_configure values('phone','rules','phone.rules');
insert into nameFind_configure values('money','rules','money.rules');
insert into nameFind_configure values('email','rules','email.rules');
insert into nameFind_configure values('percentage','rules','percentage.rules');
\install evaluatenameentityfinderpartition.zip
\install evaluatenameentityfinderrow.zip
\install findnamedentity.zip
\install trainnamedentityfinder.zip
\install nameFinderModel/date.rules
\install nameFinderModel/time.rules
\install nameFinderModel/en-sent.bin
\install nameFinderModel/email.rules
\install nameFinderModel/email.bin
\install nameFinderModel/en-ner-location.bin
\install nameFinderModel/percentage.rules
\install nameFinderModel/en-token.bin
\install nameFinderModel/names.txt
\install nameFinderModel/en-ner-organization.bin
\install nameFinderModel/money.rules
\install nameFinderModel/en-ner-person.bin
\install nameFinderModel/phone.rules
\install nameFinderModel/country.txt
```

[Table 6 - 215](#) displays the default models.

Table 6 - 215: Default models (select * from nameFind_configure;)

model_name	model_type	model_file
person	max entropy	en-ner-person.bin
location	max entropy	en-ner-location.bin
organization	max entropy	en-ner-organization.bin
date	rules	date.rules
time	rules	time.rules
phone	rules	phone.rules
money	rules	money.rules
email	rules	email.rules
percentage	rules	percentage.rules

Input

The input table should contain a text column which contains input text.

The configure_table should contain the default models.

Output

The output table contains columns specified in the OUTPUT_COLUMNS clause:

- ENTITY—Entity name
- START—Start position
- END—End position
- CONTEXT—Surrounding phrases to the entity

Example for FindNamedEntity

Table 6 - 216: Example input table: mydocs

id	src	content
1	wiki	U. S. President Barack Obama has arrived in South Korea, where he is expected to show solidarity with the president in demanding North Korea move toward ending its nuclear weapons programs.
2	wiki	Please send me email via john@teradata.com.

Example SQL-MR call

```
SELECT *
FROM FindNamedEntity(
    ON mydocs PARTITION BY ANY
    on nameFind_configure as "ConfigureTable" DIMENSION
    text_column('content')
    MODEL('all')
    OUTPUT_COLUMNS('id', 'src')
) ;
```

Table 6 - 217: Example output table

id	src	ENTITY	TYPE
2	wiki	john@teradata.com	email
1	wiki	Barack Obama	person
1	wiki	South Korea	location
1	wiki	North Korea	location

Error Messages

- If the data input is a query and there is no alias name for it, the following error message will appear:
`ERROR: all inputs of SQL-MR function FINDNAMEDENTITY must define alias names`
- If the configure table has no alias or the alias is not "ConfigureTable", the following error message will appear:
`SQL-MR function FINDNAMEDENTITY failed: configure table should have alias name ConfigureTable`
- If the format of model parameter is wrong, this error message appears:
`SQL-MR function FindNamedEntity failed: Format error for model !`
- If the specified model type is not found, this error message appears:
`SQL-MR function FindNamedEntity failed: No such model type:xxx!`
- If the specified model is not found, this error message appears:
`SQL-MR function FindNamedEntity failed: No such data model:xxx!`
- If the specified regular expression is invalid, this error message appears:
`SQL-MR function FindNamedEntity failed: Fail to compile Regular expression!`

- If the specified maximum entropy is invalid, this error message appears:
SQL-MR function FindNamedEntity failed: Fail to read model:xxx!
- If the memory of the worker node is not big enough, this error message appears:
Not enough memory to load all the models! Please contact your administrator to increase your memory on worker.XXX
- If the configure table name is not "ConfigureTable", this error message appears:
configure table should have alias name ConfigureTable
- If the configure table's schema is not correct, this error message appears:
Expected configure table types: <String, String, String>

TrainNamedEntityFinder

The NER trainer, TrainNamedEntityFinder, is based on OpenNLP, and follows its annotation. For more information on OpenNLP, see:

<http://opennlp.apache.org/documentation/1.5.2-incubating/manual/opennlp.html>

The trainer supports the English language only.

Syntax (version 1.0)

```
SELECT * FROM TrainNamedEntityFinder
(
    ON {table_name|view_name|query}
    PARTITION BY 1
    TEXT_COLUMN('text_column_name')
    ENTITY_TYPE('entity_type')
    MODEL('model_name')
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    PASSWORD('password')
    [ITERATOR('iterator')]
    [CUTOFF('cutoff')]
)
;
```

Arguments

<i>TEXT_COLUMN</i>	Required	Name of the column whose contents will be scanned. Only one column is permitted.
<i>ENTITY_TYPE</i>	Required	The entity type to be trained. The input training corpus should contain the same tag. For example, PERSON. Only one column is permitted.
<i>MODEL</i>	Required	The name of the data model file to be generated. The SQL-MR function will find the data model file in the dictionary fold or in the database.
<i>ITERATOR</i>	Optional	The iterator number for training. This is the training parameter of openNLP. The default is 100.
<i>CUTOFF</i>	Optional	The cutoff number for training. This is the training parameter of openNLP. Default is 5.
<i>DOMAIN</i>	Optional	The IP address of the queen node. The default is the queen IP.
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default is 'beehive'.
<i>USERID</i>	Optional	The Aster Database user name of the user. Default is 'beehive'.
<i>PASSWORD</i>	Required	The Aster Database password for the database user.

Input

The input table should contain a text column which contains the training data.

Output

A return value indicates success or failure in creating and installing the model. After the model is created, it is automatically installed on the Aster Database cluster.

Example for TrainNamedEntityFinder

Table 6 - 218: Example input table: myTraining

Content

```
<START:location> U. S. <END> has arrived
where he is expected to show solidarity with the country's president in demanding
<START:location> North Korea <END>
has indicated he will send an envoy to <START:location> Pyongyang <END> before the end of
the year
```

Example SQL-MR call

```
SELECT *
FROM TrainNamedEntityFinder(
    ON myTraining
    PARTITION BY 1
    type('location')
    text_column('content')
    model('location.dataset2')
    DOMAIN('153.65.197.90')
    DATABASE('sqlmr_test')
    USERID('db_superuser')
    PASSWORD('db_superuser')
) ;
```

Table 6 - 219: Example output table

Train_result

```
model installed
```

EvaluateNamedEntityFinderRow and EvaluateNamedEntityFinderPartition

The EvaluateNamedEntityFinder function takes a set of evaluating data and generates the precision, recall and f-measure value of a given maximum entropy data model.

EvaluateNamedEntityFinder functions do not support regular expression based model and dictionary based model.

EvaluateNamedEntityFinder includes two SQL-MR functions:

- *EvaluateNamedEntityFinderRow* that operates as a row function; and
- *EvaluateNamedEntityFinderPartition* that operates as a partition function.

Syntax (version 1.0)

```
SELECT * FROM EvaluateNamedEntityFinderPartition(
    ON EvaluateNamedEntityFinderRow
    (
        ON {table_name|view_name|(query)}
        TEXT_COLUMN('text_column_name')
        MODEL('model_data_file')
    )
    PARTITION BY 1
);
```

Arguments

TEXT_COLUMN Required Name of the column whose contents will be scanned. Only one column is permitted.

MODEL Required The model data file that will be evaluated. Only one model is permitted.

Input

The EvaluateNamedEntityFinderRow and EvaluateNamedEntityFinderPartition functions take as input a table with a text column of input text.

Output

Evaluation results are expressed as precision, Recall and F-measure values.

Example for EvaluateNamedEntityFinder

Table 6 - 220: Example input table: myEvaluation

id	content
1	<START:location> U. S. <END> has arrived
2	the <START:location> United States <END> are trying to coax the North back to six

Example SQL-MR call

```
SELECT * FROM EvaluateNamedEntityPartition(
    ON EvaluateNamedEntityRow(
        ON myEvaluation
        model('location.dataset2')
        text_column('content')
    )
    PARTITION BY 1
) ;
```

In this example case, the entity “United States” is not included in the training data, but the entity “U. S.” is in the training data.

Table 6 - 221: Example output

Precision	Recall	F-Measure
1	0.5	0.666666666666666

Error Messages

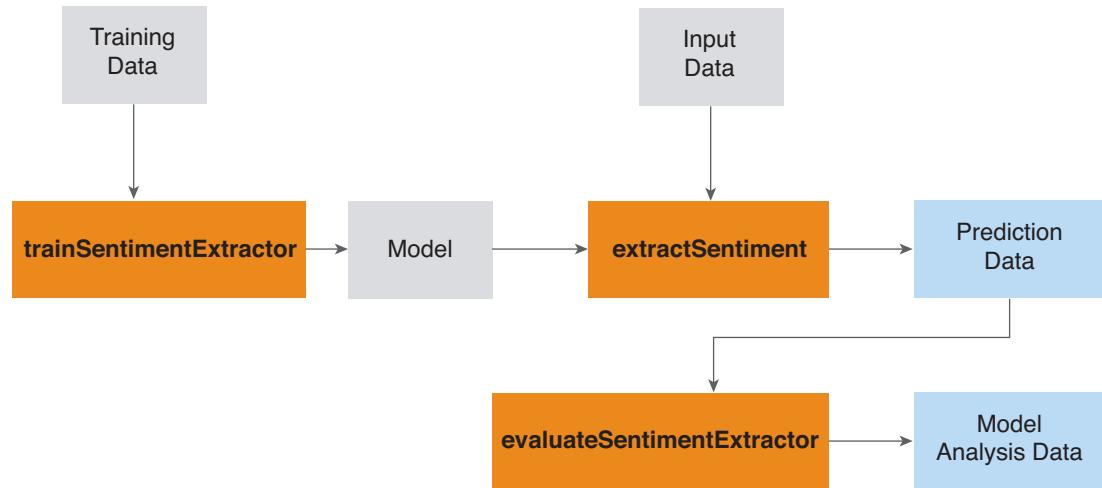
If the specified maximum entropy is invalid, the following error message will appear:

```
SQL-MR function EvaluateNamedEntityFinderRow failed: Illegal Max entropy
model!
```

Sentiment Extraction Functions

Summary

Sentiment extraction is the process of deducing a user's opinion (positive, negative, neutral) from text-based content. Sentiment extraction is useful for analyzing users' opinions as found in the content of call centers, forums, social media, and so on.



The basis for sentiment extraction in the ExtractSentiment function is either a dictionary model, or a model generated using the TrainSentimentExtractor function:

- **Dictionary model** - developed using a dictionary from WordNet. Note that the word "not", and several other negation words, have been added in to the dictionary as 999. These are listed here:
 - not
 - neither
 - never
 - no
 - scarcely
 - hardly
 - nor
 - little
 - nothing
 - seldom
 - few

Negated sentiments (for example, “I am not happy” vs. “I am not very happy” vs. “I am not at all happy”) are handled as follows:

- -1 if “not” is added,
- -1 if there is a 2 word distance between “not” and the sentiment (“happy” in these examples), and
- +1 if there is a 3 word distance.
- **MaxEnt model** - uses the model you have trained using the TrainSentimentExtractor function.

Usage

The Sentiment Extraction Functions include three SQL-MR functions as follows to extract sentiments from text content, train a model, and evaluate the results:

- [ExtractSentiment](#): a map function to extract the opinion of each document/sentence.
- [TrainSentimentExtractor](#): train a model using a classification method.
TrainSentimentExtractor supports the maximum entropy classification method.
- [EvaluateSentimentExtractor](#): evaluate the precision and recall of the ExtractSentiment function.

Permissions

Before running the ExtractSentiment, EvaluateSentimentExtractor and TrainSentimentExtractor functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

ExtractSentiment

Summary

ExtractSentiment is a map function that extracts the opinion or sentiment from input text. Much of user generated content includes the author’s feelings and opinions (happy, angry etc.). This function helps to extract the polarity of the content as positive, negative or neutral.

Background

Sentiment extraction has become more and more important over time, with the increase in user generated content being produced.

A few different sentiment extraction use cases follow:

- Support Forum - A company has an online forum where users can share knowledge and ask each other questions about how to use a particular software package. If a forum post shows the user's appreciation or involves the sharing of information, there is no need for the company's support staff to get involved in the thread. But if the forum post contains a customer's frustration at an unanswered question or the customer seems angry, then the support staff should react as soon as possible.
- Mining User Generated Reviews - There is a clothing retailer that wants to get all types of feedback for the clothing lines and accessories they sell (sizing, quality, price, style, etc.) The company has an online reviews and comments engine, and they want to get this feedback through analyzing user's reviews in the webstore, rather than using a more traditional questionnaire.
- Online Reputation Management - A large company wishes to protect its brand and reputation by monitoring negative news, blog entries, reviews and comments on the Internet.

Syntax (version 1.1)

```
SELECT *
FROM ExtractSentiment
(
    ON {table_name|view_name|(query)}
    TEXT_COLUMN('text_column_name')
    [MODEL('model_type[:model_file]')]
    [ONLY SUBJECTIVE SENTENCE('{ true | false }')]
    [ACCUMULATE('column [, ...]')]
    [LEVEL ('{DOCUMENT | SENTENCE}') ]
    [HIGH_PRIORITY('{ NEGATIVE_RECALL | NEGATIVE_PRECISION |
    POSITIVE_RECALL | POSITIVE_PRECISION | NONE }')]
    [FILTER('{ POSITIVE | NEGATIVE| ALL}') ]
)
;
```

Arguments

<i>TEXT_COLUMN</i>	Required	Name of the column whose contents will be scanned. Only one column is permitted.
<i>LEVEL</i>	Optional	The level of analysis to be performed: document or sentence. Default is 'document' level.
<i>MODEL</i>	Optional	<p>MODEL can be specified as model pairs, separated by ':'. If MODEL is not specified, the opinion word dictionary method is used. A model pair includes a model type and a model file in this format: <model_type> [:model_file]. You must first install the model file using the \install command in ACT before calling the SQL-MR function.</p> <p>Supported models include:</p> <ul style="list-style-type: none"> • DICTIONARY—use an opinion word dictionary to extract the sentiment. The dictionary model supports negation words, and these can be added to the dictionary manually. • MAX_ENTROPY—use the maximum entropy classification method to extract the sentiment from the trained model file you created with TrainSentimentExtractor.

<i>ONLY SUBJECTIVE SENTENCE</i>	Optional	Specifies whether to filter only subjective sentences, or to include objective sentences as well. Default is 'false'. Accepts 'true' or 'false': <ul style="list-style-type: none">• true—only analyze the subjective sentences. All objective sentences will be ignored.• false—both subjective sentences and objective sentences are analyzed.
<i>ACCUMULATE</i>	Optional	List of columns you want to return in the output table.
<i>HIGH_PRIORITY</i>	Optional	Sets the policy for sentiment analysis by designating the factor that has the highest priority when calculating results. In other words, some of the documents could be analyzed as having a statistically significant opinion classification, either negative or positive. But other documents may have a statistically weak classification of negative or positive. Whether the documents with a weak classification are returned, is determined by the HIGH_PRIORITY setting. Accepts the following values: <ul style="list-style-type: none">• negative recall—Includes negative results with a lower confidence level. Allows for as large a number of negative results as possible to be returned.• negative precision—Includes only the negative results that are determined to be relevant with a high degree of confidence.• positive recall—Includes positive results with a lower confidence level. Allows for as large a number of positive results as possible to be returned.• positive precision—Includes only the positive results that are determined to be relevant with a high degree of confidence.
<i>FILTER</i>	Optional	Specifies the results that will be returned. Default is 'ALL'. The value can also be: <ul style="list-style-type: none">• POSITIVE—only results with a positive sentiment are returned• NEGATIVE—only results with a negative sentiment are returned• ALL—all results are returned

Input

Before running the ExtractSentiment function, you must do the following:

- 1 Train the model using [TrainSentimentExtractor](#).
- 2 Install the model into Aster Database using the \install command in ACT.

The input table should contain a text column which contains input text.

Output

The results of the ExtractSentiment function, include:

- *out_content*: For document level, NULL will be printed. For sentence level, the sentence will be printed.
- *out_feature*: DOCUMENT or SENTENCE
- *out_polarity*: POS, NEG or NEU representing positive, negative or neutral. If the content is an empty string, UNKNOWN will be printed. If the content is NULL, output nothing.

- **out_strength:** 0, 1, or 2. A larger number means a stronger sentiment. Zero is used for neutral. 2 is stronger than 1.
- **out_sentiment_words:** This column is output only if the dictionary model is used. All the sentiment words in the input document/sentence will be returned, as determined by the model.

Example for ExtractSentiment

This example uses two user reviews of Kindle Fire, from Amazon.

Example Input Data

Create a table of Kindle Fire reviews:

```
CREATE fact table kindleView(
    id int, content varchar(1024), polarity varchar(3)
)
DISTRIBUTE BY HASH(id);
```

Insert the user review data into the table:

```
INSERT INTO kindleView values(
    1, 'I just received my Kindle Fire and I love it. I am still learning
    all the features but for me that is part of the fun. I have downloaded
    games, books, music and watched videos just like it advertised. I have
    read a lot of negative articles about the Kindle Fire and its comparison
    to the Nook and the Ipad. I would not trade my Kindle Fire for either of
    those.', 'pos');

INSERT INTO kindleView values(
    2, 'I live in Mongolia and bought Kindle Fire. Now it turns out that
    "due to my geographical location" I can not purchase/ download a single
    application, game, movie... nothing. Nada. The only thing I can do is
    download and read books. I am disappointed.', 'neg');
```

Example 1 SQL-MR call

```
SELECT * FROM ExtractSentiment
(
    ON kindleView
    text_column('content')
    model('dictionary')
    level('document')
) ;
```

Table 6 - 222: Example output table

out_content	out_feature	out_polarity	out_strength	out_sentiment_words
	DOCUMENT	neg	1	disappointed
	DOCUMENT	pos	2	love, fun, like, negative

Example 2 SQL-MR call

```
SELECT * FROM ExtractSentiment
(
    ON kindleView
    text_column('content')
    model('dictionary')
    level('sentence')
    accumulate('id')
)
;
```

Table 6 - 223: Example output table

id	out_content	out_feature	out_polarity	out_strength	out_sentiment_words
1	I just received my Kindle Fire and I love it.	SENTENCE	POS	1	love
1	I am still learning all the features but for me that is part of the fun.	SENTENCE	POS	1	fun
1	I have downloaded games, books, music and watched videos just like it advertised.	SENTENCE	POS	1	like
1	I have read a lot of negative articles about the Kindle Fire and its comparison to the Nook and the Ipad.	SENTENCE	NEG	1	negative
1	I would not trade my Kindle Fire for either of those.	SENTENCE	NEU	0	
2	I live in Mongolia and bought Kindle Fire.	SENTENCE	NEU	0	
2	Now it turns out that "due to my geographical location" I can not purchase/ download a single application, game, movie... nothing.	SENTENCE	NEU	0	
2	Nada.	SENTENCE	NEU	0	
2	The only thing I can do is download and read books.	SENTENCE	NEU	0	
2	I am disappointed.	SENTENCE	NEG	1	disappointed

Example 3 SQL-MR call

```
SELECT *
FROM ExtractSentiment
(
    ON kindleView
    text_column('content')
    model('max_entropy')
    level('document')
)
;
```

Output

Table 6 - 224: Example output table

out_content	out_feature	out_polarity	out_strength
	DOCUMENT	POS	1
	DOCUMENT	NEG	1

Error Messages

Error messages from ExtractSentiment, with explanations:

- If the function is assigned a nonexistent model, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: No model file: ...
- If the model_type of MODEL is assigned a wrong value, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: Model_type of MODEL argument can be either MAX_ENTROPY or DICTIONARY. Found: ...
- If HIGH_PRIORITY is assigned a wrong value, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: HIGH_PRIORITY argument can be NEGATIVE_RECALL, NEGATIVE_PRECISION, POSITIVE_RECALL, POSITIVE_PRECISION or NONE. Found: ...
- If FILTER is assigned a wrong value, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: FILTER argument can be POSITIVE, NEGATIVE or ALL. Found: ...
- If LEVEL is assigned a wrong value, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: LEVEL argument can be either DOCUMENT or SENTENCE. Found: ...
- If ONLY SUBJECTIVE_SENTENCE is assigned a wrong value, the following error message will appear:
SQL-MR function EXTRACTSENTIMENT failed: ONLY SUBJECTIVE_SENTENCE argument can be either 'Y' or 'N'. Found: ...

EvaluateSentimentExtractor

Summary

This partition function is used to evaluate the precision and recall of the ExtractSentiment function after training a new model or uploading a new sentiment word dictionary.

Background

Sentiment analysis is domain dependent. In other words, a new sentiment classification model or domain specific sentiment dictionary may be needed for different uses. After uploading the new model file, you can evaluate the model's efficiency by using this function.

For basic information on precision and recall calculations refer to: http://en.wikipedia.org/wiki/Precision_and_recall

Given the following definitions:

POS_EXPECT = count of expected positive sentiment in test data

NEG_EXPECT = count of expected negative sentiment in test data

NEU_EXPECT = count of expected neutral sentiment in test data

POS_TRUE = count of positive sentiment in predict and its expected is also positive

POS_RETURN = count of positive sentiment in predict, but its expected might be positive, negative or neutral.

NEG_TRUE = count of negative sentiment in predict and its expected is also negative

NEG_RETURN = count of negative sentiment in predict, but its expected might be positive, negative or neutral.

The precision and recall are calculated as:

Precision of positive sentiment = $\text{POS_TRUE} / \text{POS_RETURN}$

Recall of positive sentiment = $\text{POS_TRUE} / \text{POS_EXPECT}$

Precision of negative sentiment = $\text{NEG_TRUE} / \text{NEG_RETURN}$

Recall of negative sentiment = $\text{NEG_TRUE} / \text{NEG_EXPECT}$

Precision of all sentiment = $(\text{POS_TRUE} + \text{NEG_TRUE}) / (\text{POS_RETURN} + \text{NEG_RETURN})$

Recall of all sentiment = $(\text{POS_TRUE} + \text{NEG_TRUE}) / (\text{POS_EXPECT} + \text{NEG_EXPECT})$

If there is neutral test data, the formula can be extended following the above definitions.

Syntax (version 1.0)

```
SELECT * FROM EvaluateSentimentExtractor(
    ON {table_name|view_name|(query)}
    EXPECT_COLUMN('expect_column_name')
    RESULT_COLUMN('result_column_name')
    PARTITION BY 1
) ;
```

Arguments

EXPECT_COLUMN Required Name of the column with the expected polarity POS, NEG or NEU.

RESULT_COLUMN Required Name of the column with the result polarity POS, NEG or NEU.

Input

The input table should contain a text column which contains input text.

Output

The result of the evaluation, including precision and recall of positive sentiment, negative sentiment and both together.

Example for EvaluateSentimentExtractor

Sample Input for this example would be the trained model output by the function TrainSentimentExtractor.

Sample SQL-MR Call

```
SELECT * FROM EvaluateSentimentExtractor(
    ON ExtractSentiment(
        ON pos_train
        text_column('content')
        accumulate('category')
        model('dictionary'))
    PARTITION BY 1
    expect_column('category')
    result_column('out_polarity'));
```

Table 6 - 225: Example output table

evaluation_result
positive record (total relevant, relevant, total retrieved): 2 0 0
recall and precision: 0.00 null
negative record (total relevant, relevant, total retrieved): 0 0 0
recall and precision: null null
positive and negative record (total relevant, relevant, total retrieved): 2 0 0
recall and precision: 0.00 null

Error messages

There are no special errors, other than the Invalid Parameter error.

TrainSentimentExtractor

Summary

TrainSentimentExtractor is a reduce function to train a maximum entropy classifier for sentiment analysis. The training result will be saved as a file, and then may be installed into Aster Database using the \install command in ACT. The function has to run using PARTITION by 1.

Background

More information on maximum entropy please refer to the wiki entry at: http://en.wikipedia.org/wiki/Maximum_entropy_method

Syntax (version 1.0)

```
SELECT *
FROM TrainSentimentExtractor
(
    ON {table_name|view_name|(query)}
    PARTITION BY 1
    [TEXT_COLUMN('text_column_name')]
    [SENTIMENT_COLUMN('category_column_name')]
    MODEL_FILE('model_name')
    [ONLY SUBJECTIVE SENTENCE('{ true | false }')]
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    PASSWORD('password')
) ;
```

Arguments

<i>TEXT_COLUMN</i>	Required	The name of the column whose content will be scanned. Only one column is permitted. Default is 'content'.
<i>CATEGORY_COLUMN</i>	Required	The name of the column whose contents is the category types to be trained. Default is 'category'.
<i>CATEGORIES</i>	Required	The categories to be trained (for example, pos, neg). The categories are values of <i>CATEGORY_COLUMN</i> .
<i>MODEL_FILE</i>	Required	The name of the data model file to be generated.
<i>ONLY SUBJECTIVE SENTENCE</i>	Optional	Specifies whether to filter only subjective sentences, or to include objective sentences as well. Default is 'false'. Accepts 'true' or 'false': <ul style="list-style-type: none"> • true—only analyze the subjective sentences. All objective sentences will be ignored. • false—both subjective sentences and objective sentences are analyzed.
<i>DOMAIN</i>	Optional	IP address of the queen node. The default domain is the queen in the current cluster.
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. The default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user. The default userid is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the database user.

Input

Because TrainSentimentExtractor is a training program, the content column and category should be specified. Also the database information is required in order to install the output model.

Output

The output model with the name “model_name” will be installed in the database.

Example for TrainSentimentExtractor

Example Input Data

Create a fact table to hold the data to use to train the model:

```
CREATE FACT TABLE pos_train(
    id int,
    category varchar(10),
    content varchar(15000)
) DISTRIBUTE BY HASH(id);
```

Insert the data into the table:

```
INSERT INTO pos_train VALUES(1, 'pos', 'content1');
INSERT INTO pos_train VALUES(1, 'pos', 'content2');
```

Sample SQL-MR call

```
SELECT *
FROM TrainSentimentExtractor(
    ON (SELECT * FROM pos_train WHERE mod(id, 2)=0)
    PARTITION BY 1
    TEXT_COLUMN('content')
    SENTIMENT_COLUMN('category')
    MODEL_FILE('model1.bin')
    PASSWORD('beehive')
) ;
```

Table 6 - 226: Example output table

train_result

Model generated.

File name: model1.bin

File size(KB): 184

Training time(s): 12.5

Model successfully installed

Error Messages

If the model is assigned a nonexistent model, the following error message will appear:

```
ERROR: SQL-MR function TRAINSENTIMENTEXTRACTOR failed:
No model file: ...
```

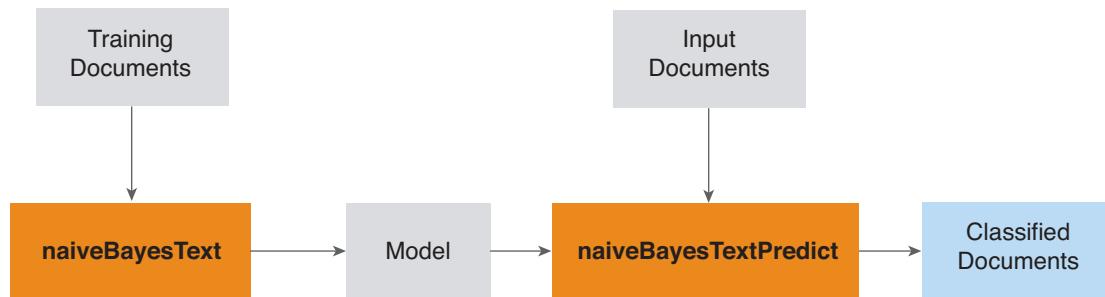
Naive Bayes Text Classifier

Summary

The SQL-MR Naive Bayes Text Classifier is a variant of the Naive Bayes classification algorithm specifically designed for document classification. This classifier executes two functions:

- `naiveBayesText`—Trains a Naive Bayes classification model on a set of documents.
- `naiveBayesTextPredict`—Uses the model generated by `naiveBayesText` to predict the category or classification of new documents.

Figure 10: SQL-MR Naive Bayes Text Classifier



Background

The Naive Bayes algorithm is a simple, yet very effective technique for classifying data. For more information about this algorithm, see “[What is Naive Bayes?](#)” on page 380.

Naive Bayes Text (`naiveBayesText`)

This function takes as input a set of classified documents (training data) and uses a variant of the Naive Bayes algorithm to generate a model that can be used by the `naiveBayesTextPredict` to classify data.

To create the model, this function breaks the training text into tokens and assigns a count value for the token for every category or classification in the training data. For example, if the training data defines two categories, *crash* and *no_crash*, this function counts the number of times the token *bag* is mentioned in text that was classified as a description of a car crash and the number of times the same token was used in text that was not a description of a car crash, as shown in the example in [Table 6 - 228](#).

However, before you can run the resulting model on the `naiveBayesTextPredict` function, you must export and reload the model into Aster Database as a CSV file using the `\install` command in ACT. For example:

```

\o sample_model.csv
COPY my_Model_Table TO STDOUT WITH DELIMITER ',';
\o
\install sample_model.csv
  
```

Input

The input table to this function contains a set of training documents that have been classified.

Syntax (version 1.0)

```
CREATE TABLE model_table_name ( PARTITION KEY(token) ) AS
SELECT token, SUM(category_1) AS category_1, ... ,
SUM(category_n) AS category_n FROM
naiveBayesText(
    ON input_table
    TEXT_COLUMN(text_column)
    CATEGORY_COLUMN(category_column)
    CATEGORIES(category_1, ..., category_n)
    [DELIMITER(''delimiter_regular_expression'')]
    [PUNCTUATION(''punctuation_regular_expression''))]
GROUP BY token;
```

Arguments

<i>model_table_name</i>	Required	The name of the model table that this function generates. This table consists of three or more columns: <i>token</i> , <i>category_1</i> , <i>category_2</i> , ..., <i>category_n</i> .
<i>category_1</i> to <i>category_n</i>	Required	The name of the token categories in the input text to include in the model. For example, crash and no_crash. Another example is normal, serious, and critical.
<i>input_table</i>	Required	The name of the input table containing the classified training documents. The table consists of three columns, a primary key column, a column for the training text data, and a column for the text categories.
<i>TEXT_COLUMN</i>	Required	The name of the column containing the training text of the documents to classify.
<i>CATEGORY_COLUMN</i>	Required	The name of the column containing the category labels of the training documents.
<i>CATEGORIES</i>	Required	A list of all the categories that are valid labels for the training documents.
<i>DELIMITER</i>	Optional	A regular expression specifying token delimiters. The default value is whitespace characters. For example: <code>DELIMITER(' [\t\f\r\n]+')</code>
<i>PUNCTUATION</i>	Optional	A regular expression specifying the punctuation to strip. The default value is periods, comma, question mark, and exclamation mark. For example: <code>PUNCTUATION ('[.,?!]')</code>

Naive Bayes Text Predict (naiveBayesTextPredict)

This function uses the model generated by the naiveBayesText function to classify or categorize the input text documents.

Syntax (version 1.0)

```
SELECT * FROM naiveBayesTextPredict (
    ON input_table
    MODEL_FILE( model_file_name )
    DOCUMENT_ID( document_id_column )
    TEXT_COLUMN( text_column )
    CATEGORIES( category_1, ... , category_n )
);
```

Arguments

<i>MODEL_FILE</i>	Required	The name of the CSV model file that was generated during the training phase, and installed using the \install command in ACT.
<i>DOCUMENT_ID</i>	Required	The name of the column containing the unique identifiers of the input text documents.
<i>TEXT_COLUMN</i>	Required	The name of the column containing the text data for the input documents.
<i>CATEGORIES</i>	Required	A list of all the categories that are valid labels for documents in the input text.
<i>DELIMITER</i>	Optional	A regular expression specifying token delimiters. The default value is whitespace characters. For example: DELIMITER(' [\t\f\r\n]+')
<i>PUNCTUATION</i>	Optional	A regular expression specifying the punctuation to strip. The default value is periods, comma, question mark, and exclamation mark. For example: PUNCTUATION ('[.,?!]')

Example

Sample input data to naiveBayesText

[Table 6 - 227](#) is a sample input table containing training data that classifies complaints as either describing car crashes or no crashes. Each row contains a single complaint, an ID that uniquely identifies the complaint, and a column for the complaint category:

Table 6 - 227: Example input table: complaints

<i>doc_id</i>	<i>text_data</i>	<i>category</i>
1	'consumer was driving approximately 45 mph hit a deer with the front bumper and then ran into a embankment head-on passneger's side air bag did deploy hit windshield and deployed outward. driver\'s side airbag cover opened but did not inflate it was still folded causing injuries.'	'crash'
2	'when vehicle was involved in a crash totalling vehicle driver\'s side/ passenger\'s side air bags did not deploy. vehicle was making a left turn and was hit by a ford f350 traveling about 35 mph on the front passenger\'s side. driver hit his head-on the steering wheel. hurt his knee and received neck and back injuries.'	'crash'
3	'consumer has experienced following problems; 1.) both lower ball joints wear out excessively; 2.) head gasket leaks; and 3.) cruise control would shut itself off while driving without foot pressing on brake pedal.'	'no_crash'

Table 6 - 227: Example input table: complaints (continued)

doc_id	text_data	category
4	'transfer case was repaired under recall. after the work was completed noise was heard intermittently. consumer took vehicle back to dealer. the dealer reinspected vehicle and informed the owner that the driveshaft was hitting the transfer case. the manufacturer has been notified.'	'no_crash'
5	'transmission would start to slip when traveling just 10mph. the rpms would be over 3 thousand. had vehicle checked at dealership & was informed transmission was stuck & that it's a factory defect almost blew up. also speedometer does not keep accurate speeds. if speed is increased it would fail to work. this was referred to mechanic by manufacturer.'	'no_crash'
6	'ue to the defective tfl ignition cable which burned the coil the vehicle stalled unexpectedly which could have resulted in a crash. also dealer replaced the r&r drive belts/speed controlable and performed vehicle tune up. please provide further information.'	'no_crash'
7	'when switch is turned on windshield wipers would not work properly. would have to jiggle switch & then wipers would move. wipers do turn off/on by themselves. recall 97v017000.'	'no_crash'
8	'consumer was driving in a rain storm when the windshield wipers stopped this happened periodically.'	'no_crash'
9	'at 66900 miles transmission has malfunctioned and will not shift into first gear. repairs were made at owner\'s expense wants reimbursement. *ml'	'no_crash'
10	'when truck was sitting on an incline it rolled on its own. manufacturer was aware of the problem. problem has not been corrected. the truck is owned by walnut hill recker manufactured in 1998.'	'no_crash'

Example SQL-MR Call

```
CREATE TABLE complaints_nb_model (PARTITION KEY(token)) AS
SELECT token, SUM(crash) AS crash, SUM(no_crash) AS no_crash FROM
naiveBayesText(
    ON complaints
    TEXT_COLUMN ('text_data')
    CATEGORY_COLUMN ('category')
    CATEGORIES ('crash', 'no_crash')
)
GROUP BY token;
```

Example Output

To display the generated model complaints_nb_model, run this query:

```
select * from complaints_nb_model order by token;
```

Table 6 - 228: Example model: complaints_nb_model

token	crash	no_crash
&	0	2
*ml	0	1
1)	0	1
10mph	0	1
1998	0	1
2)	0	1

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
3	0	1
3)	0	1
35	1	0
45	1	0
66900	0	1
97v017000	0	1
ASTER_NAIVE_BAYES_DOC_COUNTS	2	8
about	1	0
accur	0	1
after	0	1
air	2	0
airbag	1	0
almost	0	1
also	0	2
approxim	1	0
awar	0	1
back	1	1
bag	2	0
ball	0	1
been	0	2
belts/spe	0	1
blew	0	1
both	0	1
brake	0	1
bumper	1	0
burn	0	1
cabl	0	1
case	0	1
caus	1	0
check	0	1

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
coil	0	1
complet	0	1
consum	1	3
control	0	1
controlc	0	1
correct	0	1
could	0	1
cover	1	0
crash	1	1
cruis	0	1
dealer	0	2
dealership	0	1
deer	1	0
defect	0	2
deploi	2	0
did	2	0
do	0	1
doe	0	1
drive	1	3
driver	1	0
driver'	2	0
driveshaft	0	1
enbank	1	0
excessively;	0	1
expens	0	1
experienc	0	1
f350	1	0
factori	0	1
fail	0	1
first	0	1

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
fold	1	0
follow	0	1
foot	0	1
ford	1	0
front	2	0
further	0	1
gasket	0	1
gear	0	1
ha	0	4
had	0	1
happen	0	1
have	0	2
head	0	1
head-on	2	0
heard	0	1
hi	1	0
hill	0	1
hit	2	1
hurt	1	0
ignit	0	1
inclin	0	1
increasedit	0	1
inflat	1	0
inform	0	3
injuri	2	0
intermitt	0	1
involv	1	0
it	0	1
it'	0	1
itself	0	1

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
jiggl	0	1
joint	0	1
just	0	1
keep	0	1
knee	1	0
leaks;	0	1
left	1	0
lower	0	1
made	0	1
make	1	0
malfunct	0	1
manufactur	0	3
mechan	0	1
mile	0	1
move	0	1
mph	2	0
neck	1	0
nois	0	1
notfi	0	1
off	0	1
off/on	0	1
open	1	0
out	0	1
outward	1	0
over	0	1
own	0	1
owner	0	1
owner'	0	1
passenger'	1	0
passneger'	1	0

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
pedal	0	1
perform	0	1
periodc	0	1
pleas	0	1
press	0	1
problem	0	1
problems;	0	1
properli	0	1
provid	0	1
r&r	0	1
rain	0	1
ran	1	0
recal	0	2
receiv	1	0
recker	0	1
refer	0	1
reimburs	0	1
reinspect	0	1
repair	0	2
replac	0	1
result	0	1
roll	0	1
rpm	0	1
shift	0	1
shut	0	1
side	2	0
side/	1	0
sit	0	1
slip	0	1
speed	0	1

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
speedomet	0	1
stall	0	1
start	0	1
steer	1	0
still	1	0
stop	0	1
storm	0	1
stuck	0	1
switch	0	1
tfi	0	1
themselfv	0	1
thousand	0	1
took	0	1
total	1	0
transfer	0	1
transmiss	0	2
travel	1	1
truck	0	1
tune	0	1
turn	1	1
ue	0	1
under	0	1
unexpectedli	0	1
up	0	2
vehicl	1	3
walnut	0	1
want	0	1
wear	0	1
were	0	1
wheel	1	0

Table 6 - 228: Example model: complaints_nb_model (continued)

token	crash	no_crash
when	1	4
which	0	1
while	0	1
windshield	1	2
wiper	0	2
without	0	1
work	0	3
would	0	3

As mentioned previously, you must export this table to a CSV file and then feed this file to the naiveBayesTextPredict function. These commands create the CSV file:

```
-- Exports the model to a CSV file, and loads it using the
-- \install command
\o complaints_nb_model.csv
COPY complaints_nb_model TO STDOUT WITH DELIMITER ',';
\o
\install complaints_nb_model.csv
```

To verify that the complaints_nb_model.csv file has been installed, run this command in ACT:

```
=>\df
schemaName | filename | fileowner | uploadTime
-----+-----+-----+-----
public   | complaints_nb_model.csv | beehive | 2012-12-14 11:56:38.913663
...      | ...       | ...       | ...
```

Example input

[Table 6 - 229](#) is a sample input table containing text documents that need to be classified. Each row contains a document and a unique document ID:

Table 6 - 229: Example input table: 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 ALTERNATOR/ BATTERY AND STARTER, AND MODULE REPLACED 4 TIMES, BUT DEFECT STILL OCCURRING CANNOT DETERMINE WHAT IS CAUSING THE PROBLEM. *AK
2	ABS BRAKES FAIL TO OPERATE PROPERLY, AND AIR BAGS FAILED TO DEPLOY DURING A CRASH AT APPROX. 28 MPH IMPACT. MANUFACTURER NOTIFIED. *AK
3	WHILE DRIVING AT 60 MPH GAS PEDAL GOT STUCK DUE TO THE RUBBER THAT IS AROUND THE GAS PEDAL. *AK

Table 6 - 229: Example input table: complaints_test (continued)

doc_id	text_data
4	THERE IS A KNOCKING NOISE COMING FROM THE CATALYTIC CONVERTER ,AND THE VEHICLE IS STALLING. ALSO, HAS PROBLEM WITH THE STEERING. *AK
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. *AK
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. *AK
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#. *AK
8	THE AIR BAG WARNING LIGHT HAS COME ON. INDICATING AIRBAGS ARE INOPERATIVE.THEY WERE FIXED ONE AT THE TIME, BUT PROBLEM HAS REOCCURRED. *AK
9	CONSUMER WAS DRIVING WEST WHEN THE OTHER CAR WAS GOING EAST. THE OTHER CAR TURNED IN FRONT OF CONSUMER'S VEHICLE, CONSUMER HIT THE OTHER VEHICLE AND STARTED TO SPIN AROUND ,COULDN'T STOP, RESULTING IN A CRASH. UPON IMPACT, AIRBAGS DIDN'T DEPLOY. *AK
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. *AK

Example SQL-MR Call

```
SELECT * FROM naiveBayesTextPredict (
    ON complaints_test
    DOCUMENT_ID ('doc_id')
    TEXT_COLUMN ('text_data')
    CATEGORIES ('crash', 'no_crash')
    MODEL_FILE ('complaints_nb_model.csv')
);
```

Example Output

Table 6 - 230 is an example of naiveBayesTextPredict output. Each row of this table provides the classification or predicted category for each input document in the input and the log-likelihood of each of the potential categories.

Table 6 - 230: naiveBayesTextPredict output example

doc_id	prediction	loglik_crash	loglik_no_crash
2	crash	-20.0888457162399	-34.5150598682934
4	crash	-16.8586758847529	-21.6803785633408
6	crash	-34.1873553987515	-55.5260077776915
8	crash	-20.8997759324562	-32.4593348532308
10	crash	-21.5929231130161	-31.912370182849
1	crash	-39.0393856626711	-61.270612246868

Table 6 - 230: naiveBayesTextPredict output example (continued)

doc_id	prediction	loglik_crash	loglik_no_crash
3	crash	-17.8395051377646	-25.9978666768771
5	crash	-32.8531770387706	-62.5923680868503
7	crash	-30.6559524614344	-56.6009035397423
9	crash	-34.3572544355469	-57.9871979008622

TextTokenizer

Summary

The TextTokenizer function extracts tokens (or text segments) from text. Examples of tokens are words, punctuation marks, and numbers. This version of TextTokenizer supports only Chinese text.



Background

Electronic text is a linear sequence of symbols (characters, words, or phrases). To perform certain types of text analysis, the first step is to tokenize the text. Tokenization is the process of extracting the linguistic units (such as words, punctuation marks, and numbers) from text being analyzed.

Tokenization of Chinese text presents challenges not found in other languages like English. In English, words are often separated by spaces, although there are cases like “Los Angeles” and “rock 'n' roll” where the spaces do not act as word delimiters.

In the first example, “Los Angeles” is an individual concept, even though it is made up from two words separated by a space. Similarly, “rock 'n' roll” is an individual concept made up of two words, two spaces, two punctuation marks, and a single character. In addition, there are cases where a punctuation mark serves as a separator of two words as in the case of the text “I'm”, which consists of “I” and “am.”

In Chinese, sentences are represented as strings of Chinese characters or hanzi where the space and apostrophe characters, which are natural word delimiters in English, are not used. This makes the extraction of tokens more challenging when processing Chinese text. The TextTokenizer function must first identify the sequence of words in a sentence and mark their boundaries.

Syntax (version 1.0)

```
select * from TextTokenizer(
    on {table_name|view_name|(query)}
    TEXTCOLUMN('text_column_name')
    [LANGUAGE('language_type')]
    [OUTPUTDELIMITER('delimiter')]
    [OUTPUTBYWORD('true'|'false')]
    [ACCUMULATE('accumulate_column_names')]
) ;
```

Arguments

<i>TEXTCOLUMN</i>	Required	The name of the column that contains the text to be segmented.
<i>LANGUAGE</i>	Optional	The language of the input text. The possible values are: <ul style="list-style-type: none">• en (English)• zh_CN (Simple Chinese)• zh_TW (Traditional Chinese) The default value is en. However, because English tokenization is not supported in this version, this argument is required and you must specify one of the two Chinese language options (zh_CN or zh_TW).
<i>OUTPUTDELIMITER</i>	Optional	The delimiter used to separate the words in the output. The default value is "/".
<i>OUTPUTBYWORD</i>	Optional	If the value is true, the output is one line for each text; otherwise the output is per word per row. Default value is 'false'.
<i>ACCUMULATE</i>	Optional	The column names in the input table to add to the output table.

Input

- A relation that contains the text to be processed.

Output

A table containing the extracted tokens.

Example

Example Input

Table 6 - 231: Input text (cn_input)

id	txt
t1	我从小就不由自主地认为自己长大以后一定得成为一个象我父亲一样的画家，可能是父母潜移默化的影响。
t2	中华人民共和国 辽宁省 铁岭市 靠山屯 村支书 赵本山。

Example SQL-MapReduce call 1

```
select * from textTokenizer(
    on cn_input partition
    language('zh_CN')
    outputDelimiter(' ')
```

```

        outputByWord('false')
        accumulate('id')
        textColumn('txt')
    ) ;

```

Example output 1

Table 6 - 232: Output table

id	token
t1	我从小就不由自主地认为自己长大以后一定得成为一个象我父亲一样的画家,可能是父母潜移默化的影响。
t2	中华人民共和国 辽宁省 铁岭市 靠山屯村支书 赵本山。

Example SQL-MapReduce call 2

```

select * from textTokenizer(
    on cn_input partition
    language('zh_CN')
    outputByWord('true')
    accumulate('id')
    textColumn('txt')
) ;

```

Example Output 2

Table 6 - 233: Output table

id	sn	token
t1	1	我
t1	2	从小
t1	3	就
t1	4	不由自主
...
t2	1	中华人民共和国
t2	2	辽宁省
...

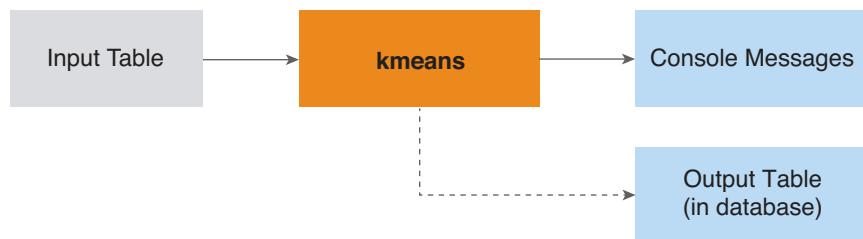
CHAPTER 7 Cluster Analysis

- [kmeans](#)
- [kmeansplot](#)
- [Minhash](#)
- [Canopy](#)

kmeans

Summary

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed *a priori*. The main idea is to define k centroids, one for each cluster. This algorithm aims at minimizing an objective function, in this case a squared error function. The objective function, which is a chosen distance measure between a data point and the cluster center, is an indicator of the distance of the n data points from their respective cluster centers.



The algorithm is composed of the following steps:

- 1 Place k points into the space represented by the objects that are being clustered. These points represent initial group centroids.
- 2 Assign each object to the group that has the closest centroid.

- 3 When all objects have been assigned, recalculate the positions of the k centroids.
- 4 Repeat steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Although it can be proved that the procedure will always terminate, the k -means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers. The k -means algorithm can be run multiple times to reduce this effect.

Background

The k -means algorithm in map-reduce consists of an iteration (until convergence) of a map and a reduce step. The map step assigns each point to a cluster. The reduce step takes all the points in each cluster and calculates the new centroid of the cluster.

Usage

Permissions

Before running the kmeans, kmeansmap, and kmeansreduce functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.0)

```
SELECT *
  FROM kmeans
  (
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    [PASSWORD('password')]
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    NUMBERK(number_of_means)
    [MEANS(starting_clusters)]
    THRESHOLD(threshold)
    MAXITERNUM(max_iterations)
  ) ;
```

Note: You must use PARTITION BY 1 when calling the kmeans SQL-MR function. This function checks for convergence of the algorithm and needs to be executed on only one vworker, so the use of PARTITION BY 1 is required.

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: <code>[::1]:2406</code> . The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: <code>DOMAIN(10.51.23.100:2406)</code>
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is <code>beehive</code> .
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default <i>USERID</i> is <code>"beehive"</code> .
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	Input table is the table containing the list of features by which we are clustering the data.
<i>OUTPUTTABLE</i>	Required	Output table is the table where output is stored. The output table contains the centroids of the clusters.
<i>NUMBERK</i>	Required if means is not present	Specifies the number of clusters to generate from the data.
<i>MEANS</i>	Required if <i>NUMBERK</i> is not present	Specifies the list of initial seed means (otherwise, a random choice is made as specified in algorithm description). Must be provided as strings of underscore delimited (_) double values, for example: <pre>means ('50_50_50_50_50_50_50_50', '150_150_150_150_150_150_150_150', '250_250_250_250_250_250_250_250', '350_350_350_350_350_350_350_350', '450_450_450_450_450_450_450_450', '550_550_550_550_550_550_550_550', '650_650_650_650_650_650_650_650', '750_750_750_750_750_750_750_750')</pre> <p>The example argument clause shown above will initialize eight clusters in eight-dimensional space. The dimensionality of the means MUST match the dimensionality of the data (that is, each mean must have <i>n</i> numbers in it, where <i>n</i> is the number of columns minus one).</p>
<i>THRESHOLD</i>	Optional	This is the convergence threshold. When the centroids move by less than this amount, the algorithm has converged. Default value is 0.0395.
<i>MAXITERNUM</i>	Optional	This is the maximum number of iterations that the algorithm will run before quitting if the convergence threshold has not been met. Default value is 10.

Input Data

This algorithm clusters *n*-dimensional numeric data (with *n* assumed to be the number of columns in the input data minus the first column, which is assumed to be the *userid/itemid*).

For example, if the required application is the clustering of points by latitude/longitude on the Earth's surface, each row would have three columns: the point-id, the latitude, and the longitude. Clustering would be performed on the latitude and longitude columns. The dimensionality *n* of the data is not specified as an argument, but implicitly derived from the data.

Output

The kmeans function outputs a message to the screen informing the user whether the function converged or not, with some additional information (see example below). The function also creates a table, whose name you specified in the OUTPUTTABLE argument, where it stores the centroids. The name of the centroids table is also given in the output to the screen.



The algorithm used by kmeans is non-deterministic, which means that if you run kmeans multiple times using the same input data, the output can be different.

Example

Example Input Data

The sample table *kmeanssample* contains:

- idnum [int]
- point1 [real]
- point2 [real]
- point3 [real]
- point4 [real]
- point5 [real]

Table 7 - 234: Example input table: kmeanssample

id	point1	point2	point3	point4	point5
1	16.21	9.07	6.19	20.93	8.74
2	18.09	14.05	10.86	6.56	11.35
3	15.56	16.61	12.30	17.11	20.54
4	13.85	6.94	17.68	14.20	20.96
5	20.19	13.77	-0.85	16.94	2.16
6	-7.86	-8.08	-4.47	-15.09	-7.11
7	-7.17	-7.89	-9.07	-8.26	-11.86
8	-7.87	-6.286	-4.21	-10.03	-14.25
9	-4.71	-10.00	-5.21	-6.31	-2.45
10	2.13	2.99	-13.33	-11.49	-9.35
11	0.96	1.18	-0.35	1.25	-0.31
12	2.72	2.08	0.12	-1.48	1.58
13	-3.70	-0.10	-1.91	0.21	1.12
14	-1.09	-3.09	1.58	-0.77	1.47
15	2.74	-0.05	-1.87	2.58	-1.96

Example SQL-MR Call

The following call will attempt to group the above 5-dimensional data points into 3 clusters:

```
SELECT *
FROM kmeans (
    ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('kmeanssample')
    outputTable('kmeanssample_centroid')
    numberK('3')
    threshold('0.01')
    maxIterNum('10')
);

```

Example Output from kMeans

Table 7 - 235: Results

message
Successful!
Algorithm converged.
Iterations: 0.
The final means are stored in the table kmeanssample_centroid, and you can use kmeansplot to assign the point to its nearest centroid.

You may then do a SELECT from the kmeanssample_centroid table to view the centroids:

```
select * from kmeanssample_centroid;
```

shows the results:

Table 7 - 236: Example output table

clusterid	means
0	0.326 0.0040 -0.486 0.358 0.38
1	16.78 12.088 9.236 15.148 12.75
2	-5.096 -5.8532 -7.258 -10.236 -9.004

Then you may use the [kmeansplot](#) function to assign old and/or new data points to the centroids that were output by the kmeans function. Note that the centroids output may be different for several runs through the function, because the initial centroids used are picked randomly.

Error Messages

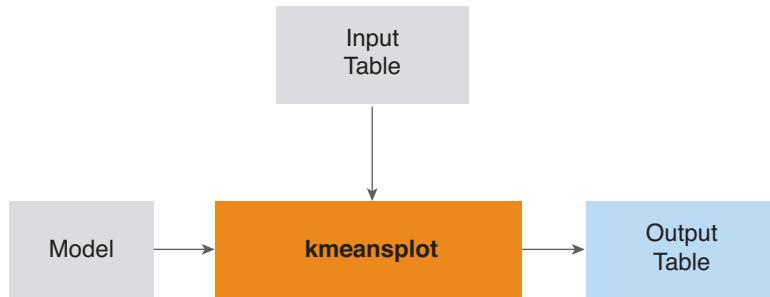
If the dimensionality of the MEANS argument is different from the dimensionality of the input data, the following error message will appear:

ERROR: SQL-MapReduce function KMEANS failed: Each mean should be of dimension [num_dimensions]

kmeansplot

Summary

After using the `kmeans` function to obtain the centroids (train the model), you can use the model to cluster new data points to these cluster centroids. The `kmeansplot` function enables you to do that.



Usage

Permissions

You must grant EXECUTE on the function “`kmeansplot`” to the database user who will run the function. Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.0)

```

SELECT *
FROM kmeansplot (
    ON inputtable PARTITION BY ANY
    ON centroids_table DIMENSION
    CENTROIDSTABLE('centroids_table')
) ;
  
```

Arguments

<i>inputtable</i>	Required	Table containing the points on which to cluster the data. If an input row in the input table contains a null value, <code>kmeansplot</code> sets the corresponding cluster ID in the output to -1.
<i>CENTROIDSTABLE</i>	Required	The table which contains the centroids trained, from the <code>kmeans</code> function.

Example

Example Input

Table 7 - 237: Example input table 1: kmeanssample

id	point1	point2	point3	point4	point5
1	16.21	9.07	6.19	20.93	8.74
2	18.09	14.05	10.86	6.56	11.35
3	15.56	16.61	12.30	17.11	20.54
4	13.85	6.94	17.68	14.20	20.96
5	20.19	13.77	-0.85	16.94	2.16
6	-7.86	-8.08	-4.47	-15.09	-7.11
7	-7.17	-7.89	-9.07	-8.26	-11.86
8	-7.87	-6.286	-4.21	-10.03	-14.25
9	-4.71	-10.00	-5.21	-6.31	-2.45
10	2.13	2.99	-13.33	-11.49	-9.35
11	0.96	1.18	-0.35	1.25	-0.31
12	2.72	2.08	0.12	-1.48	1.58
13	-3.70	-0.10	-1.91	0.21	1.12
14	-1.09	-3.09	1.58	-0.77	1.47
15	2.74	-0.05	-1.87	2.58	-1.96

Table 7 - 238: Example input table 2: kmeanssample_centroid

clusterid	means
0	0.326 0.00399999 -0.486 0.358 0.38
2	16.78 12.088 9.236 15.148 12.75
1	-5.096 -5.8532 -7.258 -10.236 -9.004

Example SQL-MapReduce Call

```
SELECT *
FROM kmeansplot (
    ON kmeanssample PARTITION BY ANY
    ON kmeanssample_centroid DIMENSION
    centroidsTable('kmeanssample_centroid')
)
ORDER BY clusterid, id;
```

Example Output from kmeansplot

Table 7 - 239: Example output table

id	clusterid	point1	point2	point3	point4	point5
1	0	16.21	9.07	6.19	20.93	8.74
2	0	18.09	14.05	10.86	6.56	11.35
5	0	20.19	13.77	-0.85	16.94	2.16
3	1	15.56	16.61	12.30	17.11	20.54
4	1	13.85	6.94	17.68	14.20	20.96
6	2	-7.86	-8.08	-4.47	-15.09	-7.11
7	2	-7.17	-7.89	-9.07	-8.26	-11.86
8	2	-7.87	-6.286	-4.21	-10.03	-14.25
9	2	-4.71	-10.00	-5.21	-6.31	-2.45
10	2	2.13	2.99	-13.33	-11.49	-9.35
11	2	0.96	1.18	-0.35	1.25	-0.31
12	2	2.72	2.08	0.12	-1.48	1.58
13	2	-3.70	-0.10	-1.91	0.21	1.12
14	2	-1.09	-3.09	1.58	-0.77	1.47
15	2	2.74	-0.05	-1.87	2.58	-1.96

Minhash

Summary

Association analysis, clustering, and the detection of similarity between items using various metrics are frequently required in data analysis, particularly over large transactional data sets.

Clustering algorithms such as the k-means algorithm and canopy partitioning perform well with physical data, but grouping items based on transaction history often requires less restrictive forms of analysis. Locality-sensitive hashing, commonly known as “minhash,” is a particularly effective way of grouping items together based on a Jaccard metric of similarity.



For example, we can declare two items to be similar because they are frequently placed in the same shopping basket by customers.

Following this approach, we can use minhash to analyze transaction data and identify clusters of “similar” items frequently bought together in a transaction. Alternatively, we might analyze the same transaction data and generate clusters of "similar" users based on the items they bought.

Background

Minhash is a probabilistic clustering method that assigns a pair of users to the same cluster with probability proportional to the overlap between the set of items that these users have bought (this relationship between users and items mimics various other transactional models). Each user u (who is a member of set U) is represented by a set of items that he has bought. The similarity between two users u_i and u_j is defined as the overlap between their item sets, given by the intersection of the item sets divided by the union of the item sets – commonly known as the “Jaccard coefficient” or “Jaccard metric.”

This similarity measure admits a locality-sensitive hashing scheme called minhash, which calculates one or more IDs for each user as the hash value (s) of a randomly chosen item from a permutation of the set of items that the user has bought. The probability that two users will be hashed to the same ID is exactly equal to their Jaccard coefficient S , as long as a class of universal hashing functions is used. To take this hashing scheme one step further, concatenating p hash-values (multiple hash values would be generated by hashing a random item from the item set with multiple hash functions) together as a distinct ID for each user makes the probability that any two users will agree on this concatenated hash key equivalent to S^p .

If each user is assigned to several ids, the odds of a collision with another id of a similar user increase. Thus, the minhash algorithm uses several hash functions, hashes a “randomly selected item” from the item set of each user (in this case the item that produces the minimum hash value for a particular hash function, hence the name of the algorithm) with each one of them, and concatenates groups of p hash values together to produce an ID, providing several ids for each user. Hence the number of key groups (p) must be a divisor of the total number of hash functions. Collisions between cluster ids lead to effective clustering.

Usage

Permissions

Before running the minhash, minhashclean, minhashmap, and minhashreduce functions, you must obtain the right to run them using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 2.0)

```
SELECT *
FROM minhash (
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
```

```

INPUTTABLE('input_table_name')
OUTPUTTABLE('output_table_name')
COLUMNNAME('column_to_be_clustered')
[SEEDTABLE('seed_table_to_use')]
[SAVESEEDTO('seed_table_to_save')]
NUMHASHFUNCTIONS('hash_function_number')
KEYGROUPS('key_group_number')
[HASH_INPUT_TYPE('bigint' | 'integer' | 'string' | 'hex')]
[MINCLUSTERSIZE('minimum_cluster_size')]
[MAXCLUSTERSIZE('maximum_cluster_size')]
) ;

```

Arguments

<i>DOMAIN</i>	Optional	IP address of the queen node. Default domain is queen of the current cluster.
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user. Default userid is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	The name of the input table. Typically it has a 'user' column and an 'items' column.
<i>OUTPUTTABLE</i>	Required	The name of the output table. This table is used to store the results.
<i>COLUMNNAME</i>	Required	The name of the input column whose values you want to hash into the same cluster.
<i>SEEDTABLE</i>	Optional	The name of the seed table whose seeds will be used for hashing. To specify this argument, the table must already exist in the database. This table is usually created from a previous run of the minhash function as specified in the 'SAVESEEDTO' argument.
<i>SAVESEEDTO</i>	Optional	The name of the table where the seeds are to be saved. You can specify this table name to save the randomly generated seeds from the current minhash run.
<i>NUMHASHFUNCTIONS</i>	Required	The calculation of the Jaccard metric (a measure of similarity between various items or user ids based upon the list of users or items, respectively, associated with them in the transaction data) involves hashing the entire list with several hash functions to calculate the minimum hash value over the list for each function. The number of hash functions to generate often determines the number of clusters generated as well as the size of the clusters generated. To find very weak similarities or relationships between various users or items, a large number of hash functions must be used.
<i>KEYGROUPS</i>	Required	The number of hash functions divided by the number of key groups must be an integer. A unique cluster id is generated by concatenating KEYGROUPS hashcodes together. A larger number of keygroups lessens the probability of collisions (hashing into the same bucket) and stunts the growth of clusters.
<i>HASH_INPUT_TYPE</i>	Optional	The input format for the list of associated items or users to be hashed. Accepts "bigint", "integer", "hex", and "string" formats.
<i>MINCLUSTERSIZE</i>	Optional	Specifies the minimum number of items or users that may be considered to constitute a cluster. Default value is 3.
<i>MAXCLUSTERSIZE</i>	Optional	Specifies the maximum size of the clusters that are under consideration. Default value is 5.

Example

Example Input Data

Input table minhash_test contains the columns, user [int] and items [varchar].

Table 7 - 240: Example input table: minhash_test

user_id	items
8	2 3 4
7	1 2 3 6 7 8
4	1 2 9 10 11
2	1 2 3 8 4 9
11	1 2 3 4 5 6
5	4 7 8 9 15
9	1 5 9 13 15
1	1 2 3 4 5 6
10	1 2 4 5 6
10	1 2 3 4 5 6
6	2 3 1 2 3 4
3	1 3 4 5 7 6
12	9 10 11 12 13

Example SQL-MapReduce call

```

SELECT *
FROM minhash (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    INPUTTABLE('minhash_test')
    OUTPUTTABLE('minhashoutput')
    COLUMNNAME('user_id')
    NUMHASHFUNCTIONS('1002')
    KEYGROUPS('3')
    HASH_INPUT_TYPE('integer')
    MINCLUSTERSIZE('3')
    MAXCLUSTERSIZE('5')
);

```

Example Output from Minhash

The following output is displayed on screen:

Table 7 - 241: Results

message
Successful.
Table 'minhashoutput' created.

The following output is saved in the 'minhashoutput' table. Each output row from the example consists of the clusterid with its space-delimited list of userids (from the left column of the input table) in the cluster.

Table 7 - 242: Example output table: minhashoutput

clusterid	userids
113642161370986921418851	10 11 2 3
15779805478707530109449421	10 11 2 3 6
13225430610307980443016007	10 11 2 4 7
10349646268200072105446942	10 11 2 6 7
106239203395173444205321496	10 11 2 7
12344657827070385230449	10 11 2 7 8
19004008338723185171484202	10 11 3
28895581518362445756670245	10 11 3 6
4924434374788909410767082	10 11 3 6 8
152898744128142219281503112	10 11 4 6 7
495150481277196637263480662	10 11 6 7
541859535161020976174101541	10 11 6 7 8
17232022929799048644790585	10 11 7
274935042133196919525694591	2 6 8
62454809812098654140821087	10 11 2
142260145203007682577362411	10 11 2 3 7
29980642978507007488743225	10 11 2 6
38306152852207335190756347	10 11 2 6 8
3288334061546975642877631	10 11 2 8
21660237640832179156454363	10 11 3 6 7

Table 7 - 242: Example output table: minhashoutput (continued)

clusterid	userids
70045548551259240278230394	10 11 3 7
186448751138694317150366830	10 11 3 9
84943064754991487180133382	10 11 6
130173924309075045247022973	10 11 6 8
143384791367401250180074909	10 11 8
1001552646572334224993302	12 2 4 5 9
26470825310212111260206159	12 4 5 9
158370363373915364183435051	2 6 7
589030277972984340180252	2 6 7 8
1649690950705728434423080	6 7 8

Error Messages

You may encounter these error messages:

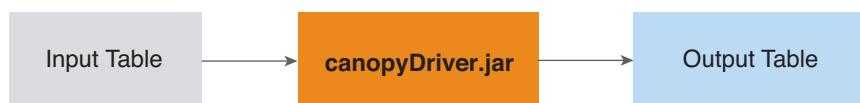
- ERROR: "numHashfunctions must be divisible by keyGroups."
- ERROR: "Please input integer cluster sizes."
- ERROR: "SQL-MR function MINHASHMAP requires argument clause: SEEDS"

Canopy

Introduction

Canopy clustering is a very simple, fast, and surprisingly accurate method for grouping objects into preliminary clusters. Each object is represented as a point in a multidimensional feature space.

The algorithm uses a fast approximate distance metric and two distance thresholds, $T_1 > T_2$, for processing. The basic algorithm begins with a set of points and identifies each point with one or more canopies – groups of interrelated, "close", or "similar" points. Any point can belong to more than one canopy (so long as the distance from the canopy center to the point is $< T_1$), and thus judicious selection of canopy centers (with none being less than T_2 apart from the next) and the points in a canopy allow for more efficient execution of clustering algorithms, which are often called within canopies.



Canopy clustering is often used as an initial step in more rigorous clustering techniques, such as k-means clustering. By starting with an initial partitioning into canopies, the number of more expensive distance measurements can be significantly reduced by ignoring points outside of the initial canopies. Also, after the initial step divides points into their respective canopies, the second step need only perform intra-canopy clustering, which can be parallelized. In other words, points that do not belong to the same canopy do not have to be considered at the same time in the clustering process.

Background

The processing is done in three map-reduce steps:

- 1 Each mapper performs canopy clustering on the points in its input set and outputs its canopies' centers (these canopies are obviously local to the mapper)
- 2 The reducer takes all the points in each (local) canopy and calculates centroids to produce the final canopy centers.
- 3 The final canopy centers are processed to eliminate centers that are too close to each other (to eliminate the effects of earlier localization).

A driver is provided that extracts information from the initial canopy generation step and uses it to make another SQL-MapReduce call that finishes the clustering process.

Installation

See “[Installing Aster Database Driver-Based Analytical Functions](#)” on page 56.

Driver Usage

```
java -classpath canopydriver.jar:<classpath to file>
      -database=<database>
      -inputtable=<inputtable>
      -outputtable=<outputtable>
      -t1=<t1>
      -t2=<t2>
      -userid=<userid>
      -password=<password>
      -domain=<domain>
```

Arguments

T1	Required This specifies the maximum distance that any point could be from a canopy center to be considered part of that canopy.
T2	Required The minimum distance that two canopy centers must be apart from each other.

Example

Example Input Data

Table 7 - 243: Example input table: canopyinput

userid	point1	point2	point3	point4
1	1.2	1.2	2.1	2
2	1.7	1.7	2.6	2.49
3	2	6	3.5	2
4	4	1.2	2.1	2.3
5	1.2	1.2	3.1	1
6	1.2	2.1	2.1	2
7	5	4.2	3.1	2

Example Query

For clarity, we have placed line breaks before each argument below. In actual usage, you would type all the arguments on a single line.

```
java
-classpath canopyDriver.jar:/home/beehive/bin/lib/sqlmr/ncluster-
sqlmr-api.jar:/home/beehive/clients_all/linux64/noarch-ncluster-jdbc-
driver.jar com.asterdata.sqlmr.analytics.clustering.canopy.canopydsriver
-database=beehive
-inputtable=canopyinput
-outputtable=canopyoutput
-t1=2
-t2=1
-domain=localhost
-userid=beehive
-password=beehive
-database=beehive
```

Example Output

These are the canopy centers.

Table 7 - 244: Example output table

canopyid	point1	point2	point3	point4
1	1.325	1.55	2.475	1.8725
2	4	1.2	2.1	2.3
3	2	6	3.5	2
4	5	4.2	3.1	2

Error Messages

You may encounter the following types of errors when you run the canopy function:

- ERROR: T1 < T2. This is impossible, and will result in empty clusters.

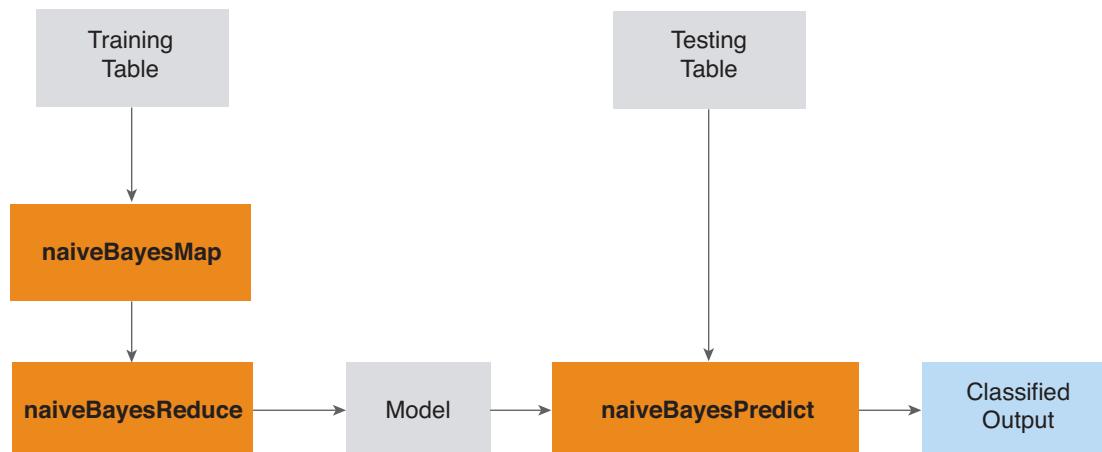
- ERROR: T1 or T2 cannot be parsed as numbers.
- ERROR: The input format must be userid, and then a n-tuple of doubles.

CHAPTER 8 Naive Bayes

- What is Naive Bayes?
- Naive Bayes Functions
- Naive Bayes Examples

What is Naive Bayes?

This is a set of functions to train a Naive Bayes classification model. The Naive Bayes algorithm is very simple, yet surprisingly effective. A training data set (for which we know discrete outcomes and either discrete or continuous input variables) is used to generate the model. The model is then used to predict the outcome of future observations, based on their input variables.



There are two main components to the Naive Bayes model:

- Bayes' Theorem

Bayes' theorem is a classical law, stating that the probability of observing an outcome given the data is proportional to the probability of observing the data given the outcome, times the prior probability of the outcome.

- the “naive” probability model

Naive Bayes

What is Naive Bayes?

The naive probability model is the assumption that the input data are independent of one another, and conditional on the outcome. This is a very strong assumption, and never true in real life, but it makes computation of all model parameters extremely simple, and violating the assumption does not hurt the model much.

Naive Bayes Functions

The classifier consists of these functions:

- `naiveBayesMap`
- `naiveBayesReduce`
- `naiveBayesPredict`

The `naiveBayesMap` and `naiveBayesReduce` functions generate a model from training data. The `naiveBayesPredict` function uses the model to make predictions on test data.

`naiveBayesMap` and `naiveBayesReduce`

Permissions

You must grant EXECUTE on the three Naive Bayes functions to the database user who will run them. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on [page 53](#).

Syntax (version 1.0)

```
CREATE TABLE model_table_name (PARTITION KEY(column_name) ) AS
SELECT * FROM naiveBayesReduce(
    ON (
        SELECT * FROM naiveBayesMap(
            ON input_table
            RESPONSE( 'response_column' )
            NUMERICINPUTS( 'numeric_input_columns' )
            CATEGORICALINPUTS( 'categorical_input_columns' )
        )
    )
    PARTITION BY column_name
) ;
```

Arguments

`RESPONSE`

Required The name of the column which contains the response variable, passed as text. The column datatype should be of type varchar, boolean, or integer.

NUMERICINPUTS	Required	At least one of the 'NUMERICINPUTS' and the 'CATEGORICALINPUTS' arguments should be specified. Any column(s) specified for 'NUMERICINPUTS' must contain numeric values. The user can either explicitly list the names of the numeric columns which will be included in the model, for example, numericinputs('input1','input2',...), or specify a range of numeric columns, for example, numericinputs('[4:33]'), or some combination of the above, for example, numericinputs('input1','[4:21],[25:53]',input73'). Ranges are specified with the following syntax: "[<start_column>:<end_column>]", with the column index starting from 0.
CATEGORICALINPUTS	Required	At least one of the 'NUMERICINPUTS' and the 'CATEGORICALINPUTS' arguments should be specified. This argument is similar to the 'NUMERICINPUTS' argument, but the column(s) specified for 'CATEGORICALINPUTS' must be varchar or integer.

Naive Bayes Examples

Example Input Data

Table 8 - 245: Example input table: nb_samples_stolenCars

id	year	color	type	origin	stolen
1	1	Red	Sports	Domestic	Yes
2	8	Red	Sports	Domestic	No
3	2	Red	Sports	Domestic	Yes
4	9	Yellow	Sports	Domestic	No
5	3	Yellow	Sports	Imported	Yes
6	10	Yellow	SUV	Imported	No
7	4	Yellow	SUV	Imported	Yes
8	11	Yellow	SUV	Domestic	No
9	12	Red	SUV	Imported	No
10	5	Red	Sports	Imported	Yes

Example SQL-MapReduce call

```

CREATE TABLE nb_stolenCars_model (PARTITION KEY(class)) AS
SELECT * FROM naiveBayesReduce(
    ON (
        SELECT * FROM naiveBayesMap(
            ON nb_samples_stolenCars
            RESPONSE('stolen')
            NUMERICINPUTS('year')
            CATEGORICALINPUTS(' [2:4] ')
)

```

```

        )
)
PARTITION BY class
) ;

```

Example Output of Naive Bayes

Table 8 - 246: Example output table: nb_stolencars_model

class	variable	type	category	cnt	sum	sumSq	totalCnt
Yes	color	CATEGORICAL	red	3			20
Yes	color	CATEGORICAL	yellow	2			20
Yes	origin	CATEGORICAL	domestic	2			20
Yes	origin	CATEGORICAL	imported	3			20
Yes	year	NUMERIC		5	15	55	20
Yes	type	CATEGORICAL	suv	1			20
Yes	type	CATEGORICAL	sports	4			20
No	color	CATEGORICAL	red	2			20
No	color	CATEGORICAL	yellow	3			20
No	origin	CATEGORICAL	domestic	3			20
No	origin	CATEGORICAL	imported	2			20
No	year	NUMERIC		5	50	510	20
No	type	CATEGORICAL	suv	3			20
No	type	CATEGORICAL	sports	2			20

naiveBayesPredict

Summary

This function uses the model generated by the naiveBayesReduce function to predict the outcomes for a test set of data.

Syntax (version 1.0)

```

SELECT *
FROM naiveBayesPredict
(
    ON input_table
    [ DOMAIN('queen_ip:port') ]
    [ DATABASE('db_name') ]
    [ USERID('user_id') ]
    PASSWORD('db_password')
    MODEL('model_table_name')
    IDCOL('test_point_id_col')

```

```

    NUMERICINPUTS ('numeric_input_columns')
    CATEGORICALINPUTS ('categorical_input_columns')
) ;

```

Parameters

<i>DOMAIN</i>	Optional	IP address of the queen node. The default domain is queen of the current cluster.
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. The default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user. The default userid is beehive.
<i>PASSWORD</i>	Required	The Aster Database user password.
<i>MODEL</i>	Required	The name of the model table generated by the naiveBayesReduce function.
<i>IDCOL</i>	Required	The name of the column that contains the id uniquely identifying test input data.
<i>NUMERICINPUTS</i>	Required	These should be the same as those in the training.
<i>CATEGORICALINPUTS</i>	Required	These should be the same as those in the training.

Example

Table 8 - 247: The input table containing the test data to be evaluated by naiveBayesPredict

id	year	color	type	origin	stolen
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

The function naiveBayesPredict uses the model generated by naiveBayesReduce to make predictions on the test data.

Table 8 - 248: Model nb_stolencars_model generated by naiveBayesReduce

class	variable	type	category	cnt	sum	sumSq	totalCnt
Yes	color	CATEGORICAL	red	3			20
Yes	color	CATEGORICAL	yellow	2			20
Yes	origin	CATEGORICAL	domestic	2			20
Yes	origin	CATEGORICAL	imported	3			20
Yes	year	NUMERIC		5	15	55	20
Yes	type	CATEGORICAL	suv	1			20
Yes	type	CATEGORICAL	sports	4			20

Table 8 - 248: Model nb_stolencars_model generated by naiveBayesReduce (continued)

class	variable	type	category	cnt	sum	sumSq	totalCnt
No	color	CATEGORICAL	red	2			20
No	color	CATEGORICAL	yellow	3			20
No	origin	CATEGORICAL	domestic	3			20
No	origin	CATEGORICAL	imported	2			20
No	year	NUMERIC		5	50	510	20
No	type	CATEGORICAL	suv	3			20
No	type	CATEGORICAL	sports	2			20

Example SQL-MapReduce call

```
SELECT * FROM naiveBayesPredict(
    ON nb_test_stolenCars
    PASSWORD('beehive')
    MODEL ('nb_stolencars_model')
    IDCOL('id')
    NUMERICINPUTS('year')
    CATEGORICALINPUTS('[2:4]')
)
ORDER BY test_id;
```

Example Output of Naive Bayes

The output provides a prediction for each data object in the test data set and specifies the log-like values used to make the predictions.

Table 8 - 249: Example output table

test_id	prediction	loglik_Yes	loglik_No
1	Yes	-3.72049098665605	-14.213638167216
2	No	-10.920490986656	-4.613638167216
3	Yes	-4.52049098665605	-20.613638167216
4	No	-9.12595609476421	-4.80817305910783
5	Yes	-3.92049098665605	-17.213638167216

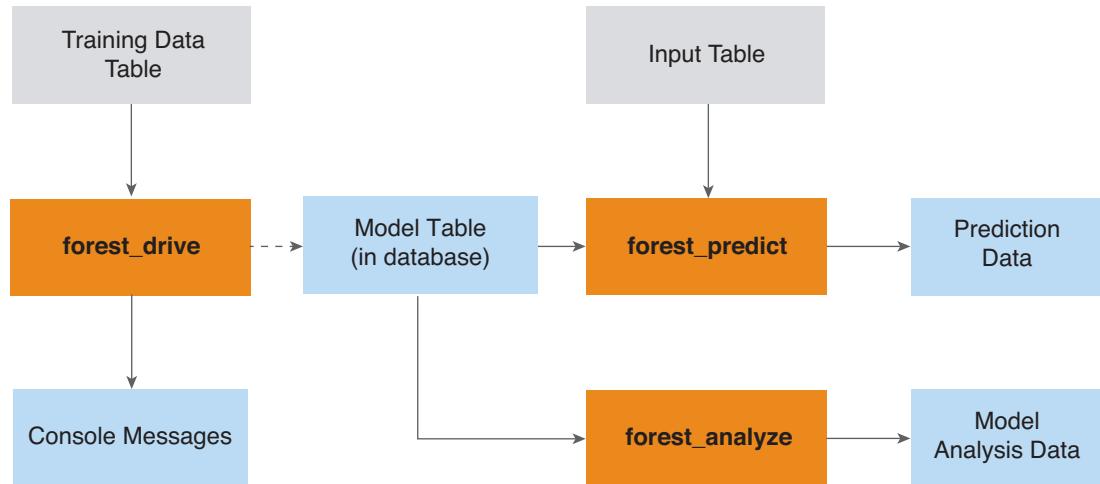
CHAPTER 9 Decision Trees

- Random Forest Functions
- Single Decision Tree Functions

Random Forest Functions

Summary

SQL-MR provides a suite of functions to create a predictive model based on a combination of the Classification and Regression Trees (CART) algorithm for training decision trees, and the ensemble learning method of bagging.



The SQL-MR decision tree functions are:

- `forest_drive`—Builds a predictive model based on training data.
- `forest_predict`—Uses the model generated by the `forest_drive` function to analyze the input data and make predictions.
- `forest_analyze`—Analyzes the model generated by the `forest_drive` function and gives weight to the variables used in the model. This helps you understand the basis by which the `forest_predict` function makes predictions.

Background

Decision trees are a supervised learning technique used for both classification and regression problems. In technical terms, a decision tree creates a piecewise constant approximation function for the training data.

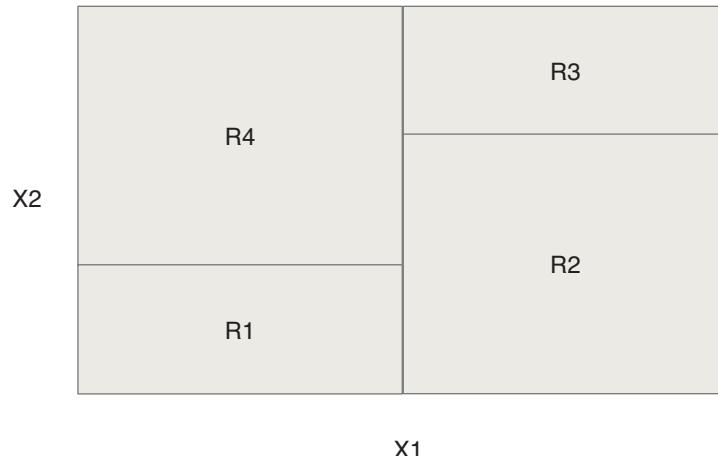
Decision trees are a common procedure used in data mining and supervised learning because of their robustness to many of the problems of real world data, such as missing values, irrelevant variables, outliers in input variables, and variable scalings. The decision tree algorithm is an *off-the-shelf* procedure, with few parameters to tune.

The SQL-MR decision tree functions implement an algorithm for decision tree training and prediction based on *Classification and Regression Trees* by Breiman, Friedman, Olshen and Stone (1984).

Decision Tree Basics

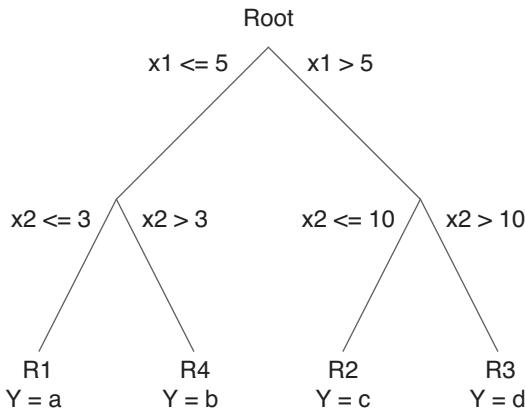
Decision trees are very simple models. For example, suppose you want to predict the value of a variable, y , and you have two predictor variables, x_1 and x_2 . You want to model y as a function of x_1 and x_2 ($y = f(x_1, x_2)$).

You can visualize x_1 and x_2 as forming a plane, and values of y at particular coordinates of (x_1, x_2) rising up out of the plane in the third dimension. A decision tree partitions the plane into rectangles and assigns each partition to predict a constant value of y , which is usually the average value of all the y values that fall in that region. You can extend this 2-dimensional example into arbitrarily many dimensions to fit models with large numbers of predictors.



In this example, the x_1 - x_2 plane has 4 regions, R1, R2, R3 and R4. The predicted value of y for any test observation that falls into R1 is the average value of y for all training observations that fall into R1.

This information can be represented in the form of a decision tree.



The algorithm starts at the Root node. If the X1 value for the data point being examined is greater than 5, then the algorithm travels down the right path. If its value of x1 is less than 5, the algorithm travels down the left path. At each subsequent node, the algorithm determines which branch to follow, until it reaches a leaf node. Then, the algorithm assigns a prediction value to the leaf node.

Advantages of Decision Trees

Decision trees are popular for many reasons. They are easy to visualize and understand, and offer an interpretable reason for the decisions they make (for example, person X is high risk because his income is below C, his total debt is above D, and his age is below A). In addition, decision trees are robust to spurious, colinear, and correlated input variables.

Disadvantages of Decision Trees

Decision tree training is a highly unstable procedure. Small differences in the training set can result in vastly different decision tree structures, and often very different outcomes. Also, because they are piecewise constant approximations, observations that fall on the boundaries between regions are more likely to have high error rates.

Usage

The SQL-MR decision tree functions create a decision model that predicts an outcome based on a set of input variables. When constructing the tree, the splitting of branches stops when any of the stopping criteria is met.

The SQL-MR decision tree functions support these predictive models:

Model	Description
Regression problems (continuous response variable)	This model is used when the predicted outcome from the data is a real number (for example, the dollar amount of insurance claims per year or the GPA expected for a college student).
Multiclass Classification (classification tree analysis)	This model is used to classify data by predicting which of the provided classes the data belongs to (for example, whether the input data is political news, economic news, or sports news).

Model	Description
Binary classification (binary response variable)	This model is used to make predictions when the outcome can be represented as a binary value (true/false, yes/no, 0/1). For example, whether the input insurance claim description data represents an accident or no accident.
<ul style="list-style-type: none">forest_driveforest_predictforest_analyze	

forest_drive

The *forest_drive* function takes as input a set training of data and uses it to generate a predictive model, which you can use later as input to the *forest_predict* function, which uses the model to make predictions.



The size of each individual decision tree generated by the *forest_drive* function should be less than 32 MB. The factors that affect the size of a decision tree are the depth of the tree, the number of categorical inputs, the number of numerical inputs, and the number of surrogates. If the size of a decision tree exceeds 32 MB, the function throws an error message. To prevent this from happening, make sure that you control the factors in the input data that increase the size of decision trees.

Syntax (version 1.0)

```
SELECT * FROM forest_drive(
    ON (SELECT 1)
    PARTITION BY 1

    [DOMAIN('host:port')]
    [DATABASE('database')]
    [USERID('user_id')]
    [PASSWORD('password')]

    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')

    RESPONSE('response_column')
    NUMERICINPUTS('numeric_input_columns')
    CATEGORICALINPUTS('categorical_input_columns')

    [TREETYPE(tree_type)]
    [NUMTREES(number_of_trees)]
    [TREESIZE(tree_size)]
    [MINNODESIZE(min_node_size)]
    [VARIANCE(variance)]
    [MAXDEPTH(max_depth)]
    [NUMSURROGATES(num_surrogates)]
) ;
```

Arguments

<i>DOMAIN</i>	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[::1]:2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
<i>DATABASE</i>	Optional	The name of the database where the input table resides. The default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default value is beehive.
<i>PASSWORD</i>	Required	The Aster Database user password.
<i>INPUTTABLE</i>	Required	The name of the table containing the input data set.
<i>OUTPUTTABLE</i>	Required	The name of the output table that this function uses to store the predictive model it generates.
<i>RESPONSE</i>	Required	The name of the column containing the response variable (that is, the quantity you want to predict).
<i>NUMERICINPUTS</i>	Either <i>NUMERICINPUTS</i> or <i>CATEGORICALINPUTS</i> is required	<p>The columns containing the numeric predictor variables. You can either explicitly list all the names, for example, <code>NUMERICINPUTS('input-col-1', 'input-col-2', ...)</code>, or specify a range of columns, for example, <code>NUMERICINPUTS('[4:33]')</code>, or some combination of the above, for example, <code>NUMERICINPUTS('input-col-1', '[4:21]', '[25:53]', 'input-col-73')</code>.</p> <p>You specify ranges as follows: <code>[start_column:end_column]</code> where <i>start_column</i> and <i>end_column</i> are the starting and ending column count numbers of the range (inclusive).</p> <p>Each count number is the column's rank in a left-to-right ranking with "1" as the leftmost column.</p> <p>Each column referred to in <i>NUMERICINPUTS</i> must contain only numeric values.</p>
<i>CATEGORICALINPUTS</i>	Either <i>NUMERICINPUTS</i> or <i>CATEGORICALINPUTS</i> is required	The columns containing the categorical predictor variables. You input data in the same way as in the <i>NUMERICINPUTS</i> clause. These columns can contain either numeric or varchar values.
<i>NUMTREES</i>	Optional	The number of trees to grow in the forest model. If not specified, the function makes an estimate and builds the minimum number of trees such that the input dataset receives full coverage.
<i>TREETYPE</i>	Optional	Specifies whether the analysis is a regression (continuous response variable) or a multiclass classification (predicting result from the number of classes). Specify <code>TREETYPE('regression')</code> or <code>TREETYPE('classification')</code> ;

TREESIZE	Optional	The number of rows each tree uses as its input data set. If not specified, the function makes an estimate and builds a tree using the minimum of: The number of rows on a vworker. or The number of rows that fit into the vworker's memory.
MINNODESIZE	Optional	Decision tree stopping criterion. The minimum size of any particular node within each decision tree. Default is 1.
VARIANCE	Optional	Decision tree stopping criterion. If the variance within any particular node dips below this value, the algorithm stops looking for splits in the branch. Default is 0.
MAXDEPTH	Optional	Decision tree stopping criterion. If the tree reaches a depth past this value, the algorithm stops looking for splits. Decision trees can grow up to $(2^{(MAXDEPTH+1)} - 1)$ nodes. Of all the stopping criteria, this has the greatest effect on the performance of the function. Default is 12.
NUMSURROGATES	Optional	Number of surrogate splits to keep for each node. Surrogate splits direct an observation to the branch of the tree it should follow if the observation has missing values for variables that are used in splits. Default is 0.

Input

Input table containing the response variable and predictor variables.

Output

The *forest_drive* function populates the table specified in the *OUTPUTTABLE* clause with the decision tree the function has created.

IMPORTANT! If a table with this name as the output table already exists in the database, the forest_drive function removes the existing table and creates a new table with the same name.

Example

This example demonstrates the use of a decision tree to predict the price class of a car model given its specifications.

Input

This SQL statement defines the training data table `carprices` ([Table 9 - 250](#)) used in this example:

```
CREATE TABLE carprices (
    car_name      varchar(100) NOT NULL,
    sports        varchar(5)  NULL,
    suv           varchar(5)  NULL,
    wagon         varchar(5)  NULL,
    minivan       varchar(5)  NULL,
    pickup        varchar(5)  NULL,
    awd           varchar(5)  NULL,
    rwd           varchar(5)  NULL,
    engine         integer    NULL,
    cylinders     smallint   NULL,
    horsepower    smallint   NULL,
    city_mpg      smallint   NULL,
    highway_mpg   smallint   NULL,
    weight         integer    NULL,
    wheelbase     smallint   NULL,
    [length]       smallint   NULL,
    [width]        smallint   NULL,
    retail_price  integer    NULL,
    dealer_price  integer    NULL,
    price_class   smallint   NULL
)
DISTRIBUTE BY HASH (weight)
STORAGE ROW;
```

For column names, [Table 9 - 250](#) uses abbreviated names so that the table can fit in this document. For example, CAR corresponds to `car_name` in the table definition SQL statement, SP to sports, and SU to suv.

Table 9 - 250: Example input table: `carprices`

CAR	SP	SU	WG	MV	PU	AW	RW	EG	CY	HP	CM	HM	WT	WB	L	W	PP	DP	PC
Acura 3.5 RL	0	0	0	0	0	0	0	3500000	6	225	18	24	3880	115	197	72	43755	39014	4
Acura 3.5 RL Navigation	0	0	0	0	0	0	0	3500000	6	225	18	24	3893	115	197	72	46100	41100	5
Acura NSX S	1	0	0	0	0	0	1	3200000	6	290	17	24	3153	100	174	71	89765	79978	7
Acura TL	0	0	0	0	0	0	0	3200000	6	270	20	28	3575	108	186	72	33195	30299	3
Audi A4 1.8T convertible	0	0	0	0	0	0	0	1800000	4	170	23	30	3638	105	180	70	35940	32506	4
Audi A4 3.0 convertible	0	0	0	0	0	0	0	3000000	6	220	20	27	3814	105	180	70	42490	38325	4
Audi A4 3.0 Q auto	0	0	0	0	0	1	0	3000000	6	220	18	25	3627	104	179	70	34480	31388	3
Audi A4 3.0 Q convertible.	0	0	0	0	0	1	0	3000000	6	220	18	25	4013	105	180	70	44240	40075	4
Audi A6 2.7 Turbo Q 4-door	0	0	0	0	0	1	0	2700000	6	250	18	25	3836	109	192	71	42840	38840	4
Audi A6 4.2 Quattro	0	0	0	0	0	1	0	4200000	8	300	17	24	4024	109	193	71	49690	44936	5
Audi A8 L Quattro	0	0	0	0	0	1	0	4200000	8	330	17	24	4399	121	204	75	69190	64740	6

Table 9 - 250: Example input table: carprices (continued)

CAR	SP	SU	WG	MV	PU	AW	RW	EG	CY	HP	CM	HM	WT	WB	L	W	PP	DP	PC
Audi S4 Avant Quattro	0	0	1	0	0	1	0	4200000	8	340	15	21	3936	104	179	70	49090	44446	5
Audi S4 Quattro	0	0	0	0	0	1	0	4200000	8	340	14	20	3825	104	179	70	48040	43556	5
Audi RS 6	1	0	0	0	0	0	0	4200000	8	450	15	22	4024	109	191	78	84600	76417	6
Audi TT 1.8	1	0	0	0	0	0	0	1800000	4	180	20	28	3131	95	159	73	35940	32512	4
Audi TT 1.8 Quattro	1	0	0	0	0	1	0	1800000	4	225	20	28	2921	96	159	73	37390	33891	4
Audi TT 3.2	1	0	0	0	0	1	0	3200000	6	250	21	29	3351	96	159	73	40590	36739	4
BMW 325Ci	0	0	0	0	0	0	1	2500000	6	184	20	29	3197	107	177	69	30795	28245	3
BMW 330Ci	0	0	0	0	0	0	1	3000000	6	225	20	30	3285	107	176	69	36995	33890	4
BMW 525i four-door	0	0	0	0	0	0	1	2500000	6	184	19	28	3428	114	191	73	39995	36620	4
BMW 530i four-door	0	0	0	0	0	0	1	3000000	6	225	20	30	3472	114	191	73	44995	41170	4
BMW 745i four-door	0	0	0	0	0	0	1	4400000	8	325	18	26	4376	118	198	75	69195	63190	6
BMW 745Li four-door	0	0	0	0	0	0	1	4400000	8	325	18	26	4464	123	204	75	73195	66830	6
BMW M3	1	0	0	0	0	0	1	3200000	6	333	16	24	3415	108	177	70	48195	44170	5
BMW X5 4.4i	0	1	0	0	0	1	0	4400000	8	325	16	22	4824	111	184	74	52195	47720	5
BMW Z4 2.5i 2-door	1	0	0	0	0	0	1	2500000	6	184	20	28	2932	98	161	70	33895	31065	3
BMW Z4 3.0i two-door	1	0	0	0	0	0	1	3000000	6	225	21	29	2998	98	161	70	41045	37575	4

Sample SQL-MapReduce Call

```

select * from forest_drive(
    on (select 1)
    partition by 1
    password('beehive')

    inputTable('carprices')
    outputTable('my_model')
    treetype('classification')

    response('price_class')
    numericInputs('cylinders','horsepower','city_mpg','highway_mpg',
                 'weight','wheelbase','length','width')
    categoricalInputs('sports','suv','wagon','minivan','pickup',
                      'awd','rwd')
    maxdepth(4)
    minnodesize(5)
    variance(0.05)
    numTrees(10)
) ;

```

Example Output of *forest_drive*

The *forest_drive* function in this example displays these messages:

Table 9 - 251: Console Messages

message
Computing 10 regression trees.
Each worker is computing 5 trees.
Each tree will contain approximately 14 points.
Poisson sampling parameter: 1.04
Query finished in 2.003 seconds.
Decision forest created in table my_model.

Also, the *forest_drive* function generates the table *my_model* which contains the generated decision-tree model. You can run this SQL statement to display the prefixes of the serialized trees in the *my_model* table:

```
select task_index, tree_num, cast(tree as varchar(50)) from my_model;
```

The output of this statement is:

Table 9 - 252: Example output table

task_index	tree_num	cast(tree as character varying(50))
0	0	{"responseCounts_": {"1514888353": 11, "-1961297549":
0	1	{"responseCounts_": {"1514888353": 3, "-1961297549": 2
0	2	{"responseCounts_": {"1514888353": 10, "-1961297549":
0	3	{"responseCounts_": {"1514888353": 9, "-1961297549": 3
0	4	{"responseCounts_": {"1514888353": 7, "-1961297549": 4
1	0	{"responseCounts_": {"1514888353": 8, "1157411398": 2,
1	1	{"responseCounts_": {"1514888353": 7, "1157411398": 2,
1	2	{"label_": "4", "size_": 12, "id_": 1, "maxDepth_": 0, "no
1	3	{"label_": "4", "size_": 10, "id_": 1, "maxDepth_": 0, "no
1	4	{"label_": "5", "size_": 8, "id_": 1, "maxDepth_": 0, "nod

forest_predict

The *forest_predict* function uses the model generated by the *forest_drive* function to generate predictions on a response variable for a test set of data.

Syntax (version 1.1)

```
SELECT *
  FROM forest_predict
```

```

(
    ON {table_name|view_name|(query)}
[DOMAIN('host:port')]
DATABASE('database_name')
USERID('user_id')
PASSWORD('password')
MODELCFILE('model_file')
FOREST('model_table')
NUMERICINPUTS('numeric_inputs')
CATEGORICALINPUTS('categorical_inputs')
IDCOL('id_column')
);

```

Arguments

<i>DOMAIN</i>	Optional	Has the form <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: <code>[: : 1] :2406</code> . The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example: <code>DOMAIN(10.51.23.100:2406)</code>
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default value is beehive.
<i>PASSWORD</i>	Required	The Aster Database user password.
<i>MODELCFILE</i>	Either MODELCFILE or FOREST must be specified	The name of the file containing the trained model generated by the <i>forest_drive</i> function. You must have installed this model previously using the ACT \install command.
<i>FOREST</i>	Either MODELCFILE or FOREST must be specified	The name of the table containing the decision forest generated by the <i>forest_drive</i> function.
<i>NUMERICINPUTS</i>	Either NUMERICINPUTS or CATEGORICALINPUTS must be specified	The columns containing the numeric predictor variables. You can either explicitly list all the names, for example, <code>NUMERICINPUTS('input-col-1','input-col-2', ...)</code> , or specify a range of columns, for example, <code>NUMERICINPUTS(' [4:33]')</code> , or some combination of the above, for example, <code>NUMERICINPUTS('input-col-1',[4:21],[25:53],'input-col-73')</code> . Use this syntax to specify ranges: <code>[start_column:end_column]</code> where <i>start_column</i> and <i>end_column</i> are the starting and ending column count numbers of the range (inclusive). Each count number is the column's rank in a left-to-right ranking with "1" as the leftmost column. Each column referred to in <i>NUMERICINPUTS</i> must contain only numeric values.
<i>CATEGORICALINPUTS</i>	Either NUMERICINPUTS or CATEGORICALINPUTS must be specified	The columns containing the categorical predictor variables. Data is input in the same way as in the NUMERICINPUTS clause. These columns can contain either numeric or varchar values.
<i>IDCOL</i>	Required	A column containing a unique identifier for each test point in the test set.

Input

The input table should contain an ID column (for example, user_id or transaction_id), so each test point can be associated with a prediction. It should also contain all columns listed in the *NUMERICINPUTS* clause (must be numeric), and all columns listed in the *CATEGORICALINPUTS* clause (can either be numeric or varchar).

To upload a model generated outside of your Aster Database, use the `/install` command in ACT. The model can be a plain text file or a ZIP file. Specify the model file using the `MODELFILE` argument when calling the function.

Output

The output table is a set of predictions for each test point.

<i>TEST_ID</i>	The unique identifier of a test point.
<i>PREDICTION</i>	The predicted value of the test point, as generated by the model.

Example

This example applies the fitted model generated by the *forest_drive* function in the previous example ([Table 9 - 250](#)) to new data to predict price class of cars based on their specifications.

Example Input Data

A table similar to [Table 9 - 250](#), but without price class information.

Example SQL-MapReduce call

```
select test_id as car_name, prediction as predicted_class
  from forest_predict(
    on carprices
    password('beehive')
    forest('my_model')
    numericInputs('cylinders','horsepower','city_mpg','highway_mpg',
                 'weight','wheelbase','length','width')
    categoricalInputs('sports','suv','wagon','minivan','pickup',
                      'awd','rwd')
    idcol('car_name')
) ;
```

Example *forest_predict* Output

Table 9 - 253: Example output table

car_name	predicted_class
Acura 3.5 RL Navigation	4
Acura TL	4
Audi A4 1.8T convertible	4
Audi A4 3.0 convertible	4
Audi A4 3.0 Quattro convertible	4

Table 9 - 253: Example output table (continued)

car_name	predicted_class
Audi A6 2.7 Turbo Quattro four-door	4
Audi A8 L Quattro	5
Audi TT 1.8 Quattro	4
Audi TT 3.2	4
BMW M3	5
BMW X5 4.4i	5
Acura 3.5 RL	4
Acura NSX S	5
Audi A4 3.0 Quattro auto	4
Audi A6 4.2 Quattro	5
Audi S4 Avant Quattro	5
Audi S4 Quattro	5
Audi RS 6	5
Audi TT 1.8	4
BMW 325Ci	4
BMW 330Ci	4
BMW 525i four-door	4
BMW 530i four-door	4
BMW 745i four-door	4
BMW 745Li four-door	4
BMW Z4 convertible 2.5i two-door	4
BMW Z4 convertible 3.0i two-door	4

forest_analyze

The *forest_analyze* function analyzes the model generated by the *forest_drive* function and gives weight to the variables used in the model. This function shows variable/attribute occurrence counts in each tree level, helping you to understand the importance of different variables in the decision making process.

Syntax (version 1.0)

```
SELECT * FROM forest_analyze(
    ON {table_name|view_name|(query)}
    [NUM_LEVELS(number_of_levels)]
) ;
```

Arguments

NUM_LEVELS Optional Number of levels to analyze. Default is 5.

Example

This example shows you how to use *forest_analyze* to analyze the sample model generated by *forest_drive* in the previous example ([Table 9 - 250](#)).

Example Input

The *my_model* table generated by the *forest_drive* function.

Example SQL-MapReduce Call

```
select task_index, tree_num, variable, level, cnt
from forest_analyze(on my_model);
```

Example *forest_analyze* Output

Table 9 - 254: Example output table

task_index	tree_num	variable	level	cnt
0	0	city_mpg	0	1
0	1	horsepower	0	1
0	2	horsepower	0	1
0	2	width	1	1
0	3	city_mpg	0	1
0	3	awd	1	1
0	4	horsepower	0	1
0	4	awd	1	1
1	0	highway_mpg	0	1
1	1	highway_mpg	0	1

Error Messages

When using this function, you may encounter these error messages:

- ERROR: Response column is not numeric.

This error message appears if the format of the *Response* column is not numeric. The values of the *RESPONSE* column must be numeric.

- ERROR: Categorical are not of type String or Integer.
This error message appears if the format of any of the *CATEGORICALINPUTS* columns is not of type String or Integer.
- ERROR: NumericInputs columns are not numeric.
This error message appears if the format of any of the *NUMERICINPUTS* columns is not numeric.
- ERROR: Invalid database parameters (*domain*, *database*, *userid*, *password*).
This error message appears if one of the parameters required to access the database is invalid.
- ERROR: Invalid tree parameters (*minNodeSize*, *variance*, *maxDepth*, *numSurrogates*).
This error message appears if one of the tree parameters is invalid.

Best Practices

Training a decision tree model is a relatively hands-off procedure, but there are several practices that you should be aware of for best performance.

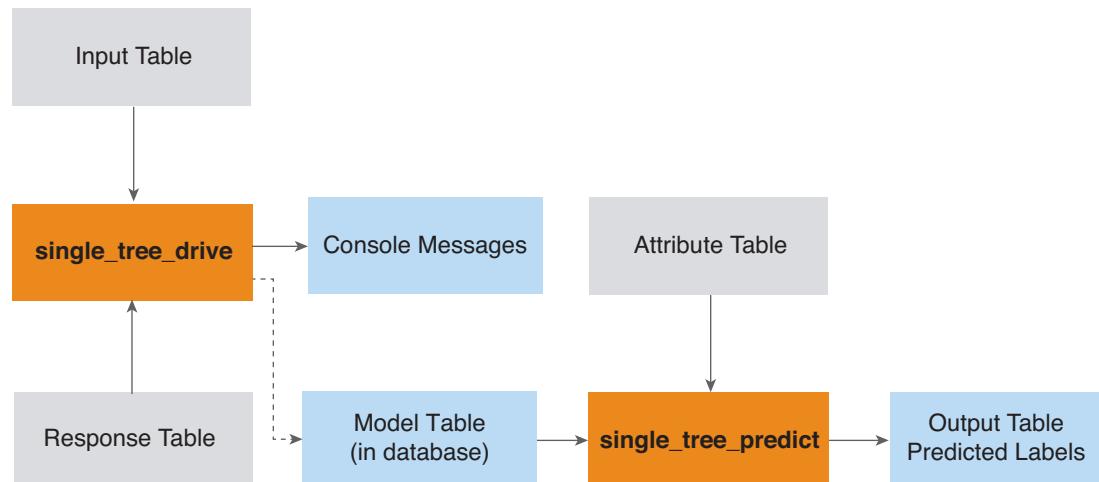
- Make sure you use the same set of columns for *CATEGORICALINPUTS* and *NUMERICINPUTS* while both building the model and using the model (for prediction); otherwise the *forest_predict* function fails.
- The *forest_drive* function computes several parameters that are important for the performance of the model, but sometimes it makes bad decisions. You can set these parameters manually to better suit your task:
 - *NUMTREES*—By default, the *forest_drive* function builds the number of trees such that the total number of sampled points is equal to the size of the original input dataset. For example, if your input dataset contains 1 billion rows, and the function determines that each tree should be trained on a sample of 1 million rows, the function decides to train 1,000 trees. Depending on your dataset, you may want more or fewer trees. As a rule of thumb, a model of 300 decision trees works well for most prediction tasks. If your dataset is small, you most likely have to specify a value for *NUMTREES*. Specify a number that is a multiple of the number of vworkers in your cluster.
 - *TREESIZE*—Each decision tree is built on a sample of the original dataset. The function computes the value of this parameter such that the decision tree algorithm does not run out of memory. With the *TREESIZE* parameter, you can specify manually how many rows each decision tree should contain. Setting this parameter too high can result in Out of Memory errors.
- You can check progress of the function in AMC. Log into the AMC and click the Processes tab. If the function is still running, you should see a function running called *forest_builder*. Click the process and click the View Logs link. The logs show stdout from the function, and give you an idea of how far along the function is. The same is true for the *forest_predict* function. Viewing the logs helps you check progress and diagnose any potential problems.

- The function does not perform well when given categorical variables with many possible values (on the order of hundreds). If you have a specific variable that can take on more than 100 values, consider consolidating some of the categories in order to improve the runtime of the function.
- The tree field in the output model table can grow to be very large. If the trees are too large, or there are too many trees in the model, the *forest_predict* function can fail, and start outputting NaNs (not a number) as predictions. Check the *forest_predict* logs in AMC to see if this is happening. If this is a problem, try one of these steps:
 - Train fewer decision trees.
 - Decrease the *MAXDEPTH* parameter in the *forest_drive* function.
 - Reduce the cardinality of your categorical input variables.
- Each vworker trains decision trees using a subsample of the data on its partition. If there is significant data skew, this can produce strange results.
- For better efficiency when running the *forest_drive* function, make sure that the input table containing the training data is randomly distributed. The best way to ensure that is by distributing the data based on a numerical attribute.

Single Decision Tree Functions

Decision trees are a common procedure used in data mining and supervised learning because of their robustness to many of the problems of real world data, such as missing values, irrelevant variables, outliers in input variables, and variable scalings. The algorithm is an “off-the-shelf” procedure, with a few parameters to tune.

This implementation creates only one decision tree as opposed to generating multiple tree in the case of random forests. This function at this point only supports classification trees on continuous variables, handles missing values during the prediction phase, supports GINI, entropy, and chi-square impurity measurements.



single_tree_drive

Summary

This function creates a single decision tree in a distributed fashion.

Background

Tree Building

The single_tree_drive function takes the entire data as training input and builds a single decision tree out of it.

Installing the Helper Functions

Make sure that these helper functions are installed before running the single_tree_drive function:

- best_splits_by_attributes
- best_splits_by_nodes
- partition_data

Difference Between the Decision Tree Approaches

In case of a random forest, each vworker operates on its own data and builds one or more decision trees. There is no need to communicate between vworkers during the forest building phase.

A forest is built based on the training data set. Once the forest is built, all the future data points are predicted against all the trees in the forest and then the function takes an aggregate for the global predicted value. In the case of a random forest, because there are many trees involved, it is not clear which variables are the most important at the different levels of the trees.

Unlike the random forest approach, single-tree approach requires vworkers to communicate with one another during the tree-building process. This communication can be very expensive depending on the number of variables and the numbers of splits being considered. To tackle this issue, the single-tree algorithm uses a sampling approach to narrow down the splits.



Tip: The single-tree implementation implements the classification tree for numeric variables.

The single_tree_drive function uses [Approximate Percentile \(approx percentile\)](#) and [Percentile](#) for sampling the split values. Also, single_tree_drive only supports categorization on numerical values; it does not support categorical inputs. The split table has all the splits for all the numerical variables to be considered for building the single decision tree.

Input Schema

Single decision trees support millions of attributes. Because the database cannot have millions of columns, you should spread the attributes across rows in the form of *key-value* pairs, where *key* is the name of the attribute and *value* is the value of the attribute.



To convert an input table used in forest_drive into an input table for single_tree_drive, use the [Unpivot](#) function.

Attribute Table

Table 9 - 255: Fact table (partition key attr_name)

Column Name	Description
<i>p-id</i>	ID of the point.
<i>attr_name</i>	Attribute name. Any attribute in the attribute table must be given a non-empty partition in the sample splits table.
<i>attr_value</i>	Attribute value. If sample values of an attribute are out of range, they cannot be used to partition the training data. In this case, this attribute is not useful. Null attribute values are viewed as missing values. Missing values are estimated by arithmetic means on an attribute basis.

Response Table

Table 9 - 256: Dimension table

Column Name	Description
<i>p-id</i>	ID of the point.
...	
<i>response_value</i>	Response value for the point.



The response table should not have a column named *node_id*.

Splits Table

Table 9 - 257: Fact table (partition key)

Column Name	Description
<i>attribute_name</i>	Attribute name.
<i>split-id</i>	Split ID.
<i>split value</i>	Split value.

Usage

Syntax (version 1.0)

```
select * from single_tree_drive(
    ON (select 1) PARTITION BY 1
    [DOMAIN('host:port')]
    [DATABASE('database')]
    [USERID('user_id')]
    PASSWORD('password')
    ATTRIBUTETABLENAME('attribute_table_name')
    RESPONSETABLENAME('response_table_name')
    OUTPUTTABLENAME('output_table_name')
    [SPLITSTABLENAME('user_provided_splits_table_name')]
    SPLITSVALUECOLUMN('splits_valcol1')
    [NUMSPLITS('num_splits_to_consider')]
    [USEAPPROXIMATESPLITS(boolean)]
    [MATERIALIZESPLITSTABLEWITHNAME('table_name_for_splits')]
    [DROPTABLE(boolean)]
    [MINNODESIZE('minimum_split_size')]
    MAX_DEPTH('max_depth')
    [IMPURITYMEASUREMENT(Gini | 'Entropy' | 'ChiSquare')]
    ATTRIBUTENAMECOLUMNS('att_col1', 'att_col2', ...)
    ATTRIBUTEVALUECOLUMN('node_col')
    RESPONSECOLUMN('response_column_name')
    IDCOLUMNS('id_column1', 'idcolumn2')
);
;
```

Arguments

<i>DOMAIN</i>	Optional	Host name, and the port of the Aster Database queen. Default host is the queen IP. Default port is the Aster Database standard port number (2406).
<i>DATABASE</i>	Optional	The name of the database where the input table exists. Default is beehive.
<i>USERID</i>	Optional	The Aster Database username of the user. Default is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>ATTRIBUTETABLENAME</i>	Required	The name of the table containing the attribute names and the values.
<i>RESPONSETABLENAME</i>	Required	The name of the table containing the response values.
<i>OUTPUTTABLENAME</i>	Required	The name of the output table which should contain the final decision tree.
<i>RESPONSECOLUMN</i>	Required	The name of the column containing the response variable in the response table.
<i>SPLITSTABLENAME</i>	Optional	The name of the table which contains the user specified splits. By default, new splits are created.

<i>SPLITSVVALUECOLUMN</i>	Required	If the user provided splits table need to be used, specify the column which contain the split value. If <i>USEAPPROXIMATESPLITS</i> is true, default is 'value', else default is user specified value for <i>ATTRIBUTEVALUECOLUMN</i> argument.
<i>NUMSPLITS</i>	Optional	We do not consider all the possible splits for all the attributes. <i>NUMSPLITS</i> specifies the number of splits to consider for each variable. Default is 10.
<i>USEAPPROXIMATESPLITS</i>	Optional	Internally, percentile values are used as the split values. Specify if exact/approximate percentiles should be used. Default is true (approximate splits).
<i>MATERIALIZESPLITSTABLE WITHNAME</i>	Optional	If you want to save the intermediate splits table, provide a name for the table. By default, the intermediate splits table is not saved.
<i>DROPTABLE</i>	Optional	When this option is set to true, if the output table name already exists, it will be dropped. Default value is false.
<i>MINNODESIZE</i>	Optional	Decision tree stopping criterion. The minimum size of any particular node within each decision tree. Default is 100.
<i>MAX_DEPTH</i>	Required	Decision tree stopping criterion. If the tree reaches a depth past this value, the algorithm stops looking for splits. Decision trees can grow up to $(2^{(MAXDEPTH+1)} - 1)$ nodes. Of all the stopping criteria, this has the greatest effect on the performance of the function. Default is 5. The maximum value is 60. If you supply a number greater than 60, an error is returned.
<i>IMPURITYMEASUREMENT</i>	Optional	The impurity measurement to be used while constructing the decision tree. We support GINI, entropy and chi-square.
<i>ATTRIBUTENAMECOLUMNS</i>	Required	Specify the list of columns in the attribute table which define the attribute.
<i>ATTRIBUTEVALUECOLUMN</i>	Required	Specify the column in the attribute table which define the value.
<i>RESPONSECOLUMN</i>	Required	Specify the column in the response table which contains the response.
<i>IDCOLUMNS</i>	Required	Specify the list of columns which specify the ID of the instance.

Example

Input

Table 9 - 258: Input table (glass_attribute_table)

PID	ATTRIBUTE	ATTRVALUE
2	RI	1.51643
14	RI	1.52121
2	Na	12.16000
14	Na	14.03000
2	Mg	3.52000
14	Mg	3.76000
2	AI	1.35000
14	AI	0.58000
2	Si	72.89000

Table 9 - 258: Input table (glass_attribute_table) (continued)

PID	ATTRIBUTE	ATTRVALUE
14	Si	71.79000
2	K	0.57000
14	K	0.11000
2	Ca	8.53000
14	Ca	9.65000
2	Ba	0.00000
14	Ba	0.00000
2	Fe	0.00000
14	Fe	0.00000
5	RI	1.53393
5	Na	12.30000
5	Mg	0.00000
5	AI	1.00000
...

Response Table

Table 9 - 259: Response table (glass_response_table)

PID	RESPONSE
6	2
18	2
30	3
26	2
25	5
1	1
13	1
5	2
12	2
24	1
3	1
15	3

Table 9 - 259: Response table (glass_response_table) (continued)

PID	RESPONSE
27	2
8	1
20	2
17	7
29	2
2	3
14	3
11	2
23	2
10	2
22	2
7	3
9	7
21	1
4	6
16	2
...	...

Example SQL-MapReduce Call

```
SELECT * FROM single_tree_drive(
    ON (SELECT 1)
    PARTITION BY 1
    ATTRIBUTETABLENAME('glass_attribute_table')
    OUTPUTTABLENAME('glass_attribute_table_output')
    MATERIALIZESPLITSTABLEWITHNAME('splits_small')
    RESPONSETABLENAME('glass_response_table')
    NUMSPLITS('3')
    IMPURITYMEASUREMENT('gini')
    MAX_DEPTH('10')
    IDCOLUMNS('pid')
    ATTRIBUTENAMECOLUMNS('attribute')
    ATTRIBUTEVALUECOLUMN('attrvalue')
    RESPONSECOLUMN('response')
    MINNODESIZE('3')
    PASSWORD('beehive')
    USEAPPROXIMATESPLITS('false'));
```

Output

Table 9 - 260: Output message

message
Model for input tables:"glass_attribute_table", "new_glass_response_table" created in table: "glass_attribute_table_output"
Depth of the tree is:8
(2 rows)

Use this SQL statement to display the output table:

```
select * from glass_attribute_table_output order by node_id;
```

Table 9 - 261: Output table (glass_attribute_table_output)—columns 1–7

node_id	node_size	node_gini(p)	node_entropy	node_chisq_pv	node_label	node_majorvotes
0	30	0.728888888888889	2.1742232850498	1	2	13
1	11	0.446280991735537	1.27761343681912	1	2	8
2	19	0.786703601108033	2.3545502265469	1	2	5
3	5	0.56	1.37095059445467	1	2	3
4	6	0.2777777777777778	0.650022421648354	1	2	5
5	6	0.6666666666666667	1.79248125036058	1	7	3
6	13	0.662721893491124	1.57662122010749	1	2	5
8	4	0.375	0.811278124459133	1	2	3
11	4	0.375	0.811278124459133	1	7	3
13	10	0.62	1.48547529722733	1	2	5
14	3	0.444444444444444	0.91829583405449	1	1	2
27	4	0.5	1	1	3	2
28	6	0.611111111111111	1.45914791702725	1	2	3
58	5	0.48	0.970950594454668	1	2	3
117	4	0.5	1	1	1	2
235	3	0.444444444444444	0.91829583405449	1	1	2

Table 9 - 262: Output table (glass_attribute_table_output)—columns 8–15

split_value	split_gini(p)	split_entropy	split_chisq_pv	left_id	left_size	left_label	left_majorvotes
12.96	0.661881977671451	1.95967340364671	0.181990229855887	1	11	2	8
0.11	0.406060606060606	0.977717045651225	0.328959086262671	3	5	2	3
2.96	0.663967611336032	1.64478754545057	0.00916424471287824	5	6	7	3
1.3	0.3	0.649022499567306	0.0820849986238987	7	1	1	1
1.54	0.1666666666666667	0.333333333333333	0.121335250358482	9	4	2	4
0.45	0.4166666666666667	0.874185416306088	0.111610225094714	11	4	7	3
8.96	0.57948717948718	1.35458772880283	0.207871883589572	13	10	2	5
8.43	0.25	0.5	0.248213078989924	17	2	2	2
8.43	0	0	0.0455002638963578	23	1	6	1
1.51683	0.5666666666666667	1.27548875021635	0.329192987807906	27	4	2	2
13.33	0	0	0.0832645166635491	29	2	1	2
0.45	0	0	0.0455002638963578	55	2	2	2
0.45	0.4	0.80912549537889	0.049787068367864	57	1	3	1
1.54	0.4	0.8	0.361310428526179	117	4	2	2
8.43	0.333333333333333	0.688721875540867	0.248213078989924	235	3	1	2
1.3	0.333333333333333	0.6666666666666667	0.386476230771233	471	1	1	1

Table 9 - 263: Output table (glass_attribute_table_output)—columns 16–20

right_id	right_size	right_label	right_majorvotes	attribute
2	19	2	5	Na
4	6	2	5	Fe
6	13	2	5	Mg
8	4	2	3	Al
10	2	2	1	Al
12	2	5	1	K
14	3	1	2	Ca
18	2	2	1	Ca
24	3	7	3	Ca
28	6	2	3	RI
30	1	3	1	Na
56	2	3	2	K
58	5	2	3	K

Table 9 - 263: Output table (glass_attribute_table_output)—columns 16–20 (continued)

right_id	right_size	right_label	right_majorvotes	attribute
118	1	2	1	AI
236	1	2	1	Ca
472	2	1	1	AI

single_tree_predict

This function is used for applying a tree model to a data input, outputting predicted labels for each data point.

Usage

Syntax (version 1.1)

```
SELECT * FROM single_tree_predict (
    ON attribute_table as attribute_table partition by pid_col1[, pid_col2[, ...]]
    ON model_table as model_table DIMENSION
    AttrTable_GroupbyColumns('gcol1'[, 'gcol2'...])
    AttrTable_pidColumns('pid_col1'[, 'pid_col2'...])
    AttrTable_valColumn('value_column')
    ModelTable_nodeColumn('node_column')
    ModelTable_sizeColumn('size_column')
    ModelTable_leftSizeColumn('left_size_column')
    ModelTable_rightSizeColumn('right_size_column')
    ModelTable_attrColumns('attr_1'[, 'attr_2'[, ...]])
    ModelTable_splitColumn('split_column')
    ModelTable_labelColumn('label_column')
    ModelTable_leftLabelColumn('left_label_column')
    ModelTable_rightLabelColumn('right_label_column'))
;
```

Arguments

<i>attribute_table</i>	Required	The input data to be applied on.
<i>AttrTable_GroupbyColumns</i>	Required	Arguments of this argument give out the columns that <i>attribute_table</i> is partitioned on. Each partition contains one particular attribute of the input data.
<i>AttrTable_pidColumns</i>	Required	Gives out the columns that define the point ids.
<i>AttrTable_valColumn</i>	Required	Indicates the column that contains the input values.
<i>model_table</i>	Required	The tree model. Each row represents a tree node.

A typical model input schema looks like this:

Table 9 - 264: Input Schema

Column	Type
node_id	integer
node_size	integer
node_impurity	double precision
node_label	character varying
node_majorvotes	integer
split_value	double precision
split_impurity	double precision
left_id	integer
left_size	integer
left_label	character varying
left_majorvotes	integer
right_id	integer
right_size	integer
right_label	character varying
right_majorvotes	integer
attribute	character varying

The columns to be used in the model table are indicated by the following argument clauses:

Table 9 - 265: Column descriptions

Column	Description
ModelTable_nodeColumn	The node id column
ModelTable_labelColumn	Labels on each node
ModelTable_sizeColumn	The node size column
ModelTable_leftSizeColumn	Sizes of the left children to each node
ModelTable_rightSizeColumn	Sizes of the right children to each node
ModelTable_attrColumns	Columns that define which attribute a split belongs to
ModelTable_splitColumn	Split values
ModelTable_leftLabelColumn	Labels of the left children to each node
ModelTable_rightLabelColumn	Labels of the right children to each node

Output Schema

Table 9 - 266: Output schema

Column	Description
<pid_col1>, <pid_col2>, ...	The output schema contains one or more pid columns whose definition are the same as the pid columns in the input attribute table.
<i>pred_label</i>	The predicted labels for each input data point.

Example

Input

This function takes two input tables:

- `glass_attribute_table_output`—This is the model outputted by `single_tree_drive`.
- `glass_attribute_table`—This is the table containing the labels to be predicted. For simplicity, this example, uses the same table used as input to the `single_tree_drive` function example ([Table 9 - 258](#)).

Sample SQL-MapReduce Call

```
SELECT * FROM single_tree_predict (
    ON glass_attribute_table as attribute_table partition by pid
        order by attribute
    ON glass_attribute_table_output as model_table DIMENSION
        ATTRTABLE_GROUPBYCOLUMNS('attribute')
        AttrTable_pidColumns('pid')
        AttrTable_valColumn('attrvalue')
        ModelTable_nodeColumn('node_id')
        ModelTable_sizeColumn('node_size')
        ModelTable_leftSizeColumn('left_size')
        ModelTable_rightSizeColumn('right_size')
        ModelTable_attrColumns('attribute')
        ModelTable_splitColumn('split_value')
        ModelTable_labelColumn('node_label')
        ModelTable_leftLabelColumn('left_label')
        ModelTable_rightLabelColumn('right_label')
) ORDER BY pid;
```

Output

Table 9 - 267: Output table

pid	<i>pred_label</i>
1	1
2	2
3	1
4	6
5	2

Table 9 - 267: Output table (continued)

pid	pred_label
6	2
7	3
8	5
9	7
10	2
11	2
12	2
13	1
14	3
15	3
16	2
17	7
18	2
19	5
20	2
21	1
22	2
23	2
24	1
25	2
26	2
27	2
28	7
29	1
30	3

CHAPTER 10 Association Analysis

- Basket Generator (`basket_generator`)
- Collaborative Filtering (`cfilter`)

Basket Generator (`basket_generator`)

Summary

This function generates sets (“baskets”) of items. The input is typically a set of purchase transaction records or web pageview logs. Each basket is a unique combination or permutation of items. You specify the desired basket size. Combinations and Permutations are returned in lexicographical order.

The resulting baskets can be used as part of a collaborative filtering algorithm. This is useful for analyzing purchase behavior of users in-store or on a website. This function can also operate on activity data (for example, “users who viewed this page also viewed this page”).



Background

Retailers mine transaction data to track purchasing behavior or viewing behavior. A retailer's goal is to find interesting combinations (called baskets) of items purchased together or shopped for at the same time. A frequent need is to automatically identify interesting baskets and also look for trends over time and compare other attributes (for example, compare stores). Having a general function that can operate on data structured in the form often present for retail will make interesting market basket analysis possible.

This general function is intended to help facilitate market basket analysis by operating on data that is structured in a form typical of retail transaction history databases.

Usage

Syntax (version 1.1)

```
SELECT *
  FROM basket_generator
  (
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    [BASKET_SIZE('basket_size_value')]
    BASKET_ITEM('basket_item_column')
    ACCUMULATE('column1 [, column2, ...]')
    COMBINATIONS('true' | 'false')
    [ITEM_SET_MAX('item_set_max_value')]
  );
```

Arguments

PARTITION BY	Required	Must specify the column(s) to partition by. This specifies the sets for which combinations will be generated and summed. Make sure you do not use quotes in the PARTITION BY clause. If you use quotes, this will be treated as string, which is a constant expression, and all the data would be grouped to one partition.
BASKET_ITEM	Required	Name(s) of the input column(s) that contains the items to be collected into baskets. If more than one input column is specified, every unique combination of input column values is treated as one item. For example, if a single column is used, this is often the column containing the SKU that identifies an item that was sold. If you wanted to further break down the results, you could specify both the SKU column and the month manufactured, color and/or size columns.
BASKET_SIZE	Required	Integer number of items to be included in a basket. The default is two items.
ACCUMULATE	Optional	Names of input columns that will be returned as-is in the output. If supplied, these must be columns that are part of the ON relation. All input columns not named here are left out of the output. Columns in the ACCUMULATE clause should be a subset of the columns in the PARTITION BY clause. Otherwise the deterministic property is not guaranteed.
COMBINATIONS	Optional	Defaults to 'true'. If 'true' the function returns a basket for each unique <i>combination</i> of items. If 'false' the function returns a basket for each unique <i>permutation</i> of items. Combinations are returned in lexicographical order. For a <i>combination</i> , the order of the items doesn't matter (the basket "tomatoes and basil" is considered to be the same basket as "basil and tomatoes"). For a <i>permutation</i> , every unique ordering of the items constitutes a unique basket.
ITEM_SET_MAX	Optional	(Type=int) [default=100]. This is the maximum number of items to be considered in a partition. If the number of items in any partition exceeds ITEM_SET_MAX, no combinations (or permutations) will be emitted for that partition.

Notes

If the number of combinations (or permutations) exceeds one million, no rows are emitted. The maximum possible number of combinations or permutations you might generate will depend on:

- n , the number of distinct items that may appear in a basket (in other words, the cardinality of the BASKET_ITEM column(s)), and
- r , the BASKET_SIZE.

Number of combinations generated = nC_r

which we can also express as $\frac{n!}{r!(n-r)!}$

Number of permutations generated = nP_r

which we can also express as $\frac{n!}{(n-r)!}$

Output

Returns whatever is specified in ACCUMULATE along with one column for each of the “basket_items” specified.

Examples

Below are two examples of the use of the function to generate market baskets along with other columns of interest.

Example Input Data

The input data, stored in table *transactions*, has the following columns:

Table 10 - 268: Example input table: transactions

storeid	sku	regid	tran_no
123	san carlos	32	1234435
123	san jose	45	234
123	san francisco	213	2345
123	new york	381	-819752
123	dallas	549	-1435797
123	LA	717	-2051842
123	seattle	885	-2667887
123	las vegas	1053	-3283932
123	Mexico	1221	-3899977
123	washington	1389	-4516022
124	san carlos	1526	-5132067
124	san jose	32	-5748112
124	san francisco	45	-6364157
124	new york	213	-6980202
124	dallas	381	-7596247
124	LA	549	-8212292
124	seattle	717	-8828337
124	las vegas	885	-9444382

Table 10 - 268: Example input table: transactions (continued)

storeid	sku	regid	tran_no
124	Mexico	1053	-10060427
124	washington	1221	-10676472
125	san carlos	1389	-11292517
125	san jose	1526	-11908562
126	san francisco	1243	-12524607

Example SQL-MapReduce call (1 of 2)

```
SELECT storeid, sku1, sku2, sku3, count(1)
FROM basket_generator
(
  ON transactions
  PARTITION BY storeid, regid, tran_no
  BASKET_SIZE(3)
  BASKET_ITEM('sku')
  ACCUMULATE('storeid')
  ITEM_SET_MAX(200)
) GROUP BY 1,2,3,4;
```

Example Output from Basket Generator (1 of 2)

Table 10 - 269: Example output table (1 of 2)

storeid	sku1	sku2	sku3	count(1)
1	83	97	1213	123
2	78	122	7812	88

Example SQL-MapReduce call (2 of 2)

```
SELECT storeid, EXTRACT(month FROM dt) AS mnth, sku1, sku2, count(1)
FROM basket_generator
(
  ON transactions
  PARTITION BY storeid, dt, regid, tran_no
  BASKET_SIZE(2)
  BASKET_ITEM('sku')
  ACCUMULATE('storeid', 'dt')
  ITEM_SET_MAX(200)
) GROUP BY 1,2,3,4;
```

Example output from Basket Generator (2 of 2)

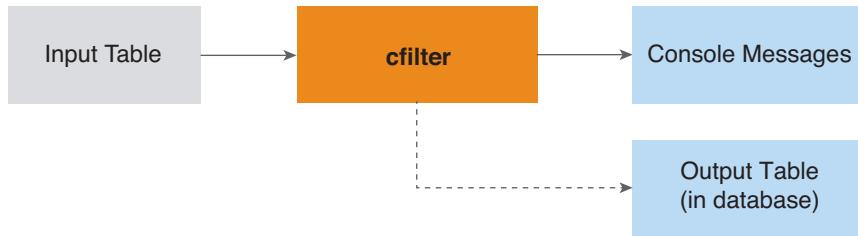
Table 10 - 270: Example output table (2 of 2)

storeid	mnth	sku1	sku2	cnt
1	2	83	1213	7824
3	4	122	7812	3112

Collaborative Filtering (cfILTER)

Summary

This function performs collaborative filtering via a series of SQL commands and SQL-MapReduce functions. You run this function via an internal JDBC wrapper function.



Background

Collaborative filtering is used by analysts to find items or events that are frequently paired with other items or events. For example, the Amazon.com feature, “People who shopped for this item also shopped for...” uses a collaborative filtering algorithm. Another use is “People who viewed this profile also viewed this profile” on LinkedIn. Aster Database’s collaborative filter (cfILTER) is a general-purpose tool that can provide answers in many similar-use cases.

Usage

Syntax (version 1.4)

```

SELECT * FROM cfILTER (
    ON (SELECT 1)
    PARTITION BY 1
    [ domain('ip_address') ]
    [ database('db_name') ]
    [ userid('user_id') ]
    password('password')
    inputTable('input_table_name')
    outputTable('output_table_name')
    inputColumns('source_column1', 'source_column2',...)
    joinColumns('join_column1', 'join_column2',...)
    [ otherColumns('other_column1', 'other_column2',...) ]
    [ partitionKeyColumn ('partitionKeyColumn1') ]
    [ maxSet('max_item_set') ]
    [ dropTable('true' | 'false') ]
);
  
```

Arguments

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [::1]:2406. The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default USERID is "beehive".
<i>PASSWORD</i>	Required	The Aster Database password of the user.
<i>INPUTTABLE</i>	Required	Name of the input table containing the data to be filtered.
<i>OUTPUTTABLE</i>	Required	Name of the output table into which the function writes the final results. If the output table already exists, then you should also pass the DROPTABLE ('true') argument, to drop it before writing the new results. Otherwise, an exception is thrown. The output table contains the columns listed in the Output table below.
<i>INPUTCOLUMNS</i>	Required	A list of input columns to filter. The column names are single-quoted and written in the comma-delimited format <' <i>col1</i> ', ' <i>col2</i> ', ...>.
<i>JOINCOLUMNS</i>	Required	A list of columns to join on. The column names are single-quoted and written in the comma-delimited format <' <i>col1</i> ', ' <i>col2</i> ', ...>.
<i>OTHERCOLUMNS</i>	Optional	A list of other columns to output. These will pass through the function unchanged. The column names are single-quoted and written in the comma-delimited format <' <i>col1</i> ', ' <i>col2</i> ', ...>.
<i>PARTITIONKEY</i>	Optional	Single column used as partition key for the newly created output table. Default <i>partitionKey</i> is <i>col1_item1</i> .
<i>MAXSET</i>	Optional	Size of the maximum item set to be considered. Default <i>maxSet</i> is 100.
<i>DROPTABLE</i>	Optional	If you set this option to true, if the output table name already exists, the function drops the table. Default value is false.

Output

The output table contains these columns:

Table 10 - 271: Example output table

Column Name	Description
Col1_item1	Name of item1.
Col2_item2	Name of item2.
Cntb	Count of the co-occurrence of both items (situations when people buy the two items together).
Cnt1	Count of the occurrence of item1.
Cnt2	Count of the occurrence of item2.
Score	The product of two conditional probabilities: $(cntb * cntb) / (cnt1 * cnt2)$, or $P(\text{item2} \text{item1}) * P(\text{item1} \text{item2})$

Table 10 - 271: Example output table (continued)

Column Name	Description
Support	How often this pattern occurs: $cntb/tran_cnt$ $tran_cnt$ is the total number of transactions. Among all the transactions, the percentage of the items co-occurrence.
Confidence	The level of confidence about $\{item1 \Rightarrow item2\}$. $cntb/cnt1$ The percentage of item2 occurrence in all the transactions in which item1 occurs.
Lift	The ratio of the observed support value to the expected support value if item1 and item2 were independent. $(cntb/tran_cnt) / [(cnt1/tran_cnt) * (cnt2/tran_cnt)]$, or $support(item1 \text{ and } item2) / [support(item1) * support(item2)]$.
Lift > 1	The occurrence of item1 or item2 has a positive effect on the occurrence of the other items.
Lift < 1	The occurrence of item1 or item2 has a negative effect on the occurrence of the other items.
Lift = 1	The occurrence of item1 or item2 has a no effect on the occurrence of the other items.
Z_score	Assuming $cntb$ follows a normal distribution, the z_score is $(cntb - mean(cntb))/sd(cntb)$. It is a way to measure how significant the co-occurrence is. If all $cntb$ (s) are the same, then $sd(cntb)$ is 0, and the zscore will not be computed.

Example

Example Input Data

Table 10 - 272: Example input table: cfILTER_TEST

tranid	dt	storeid	region	item	sku	category
1	20100715	1	west	milk	1	dairy
1	20100715	1	west	butter	2	dairy
1	20100715	1	west	eggs	3	dairy
1	19990715	1	west	flour	4	baking
3	20100715	1	west	milk	1	dairy
3	20100715	1	west	eggs	3	dairy
3	19990715	1	west	flour	4	baking
5	20100715	2	west	butter	2	dairy

Table 10 - 272: Example input table: cfilter_test (continued)

tranid	dt	storeid	region	item	sku	category
5	20100715	2	west	eggs	3	dairy
5	19990715	2	west	flour	4	baking
7	20100715	2	west	eggs	3	dairy
7	19990715	2	west	flour	4	baking
2	20100715	1	west	milk	1	dairy
2	20100715	1	west	butter	2	dairy
2	20100715	1	west	eggs	3	dairy
4	20100715	1	west	milk	1	dairy
4	20100715	1	west	butter	2	dairy
6	20100715	2	west	milk	1	dairy
6	20100715	2	west	eggs	3	dairy
8	20100715	3	west	butter	2	dairy
8	20100715	3	west	eggs	3	dairy
8	19990715	3	west	flour	4	baking

Example SQL-MapReduce Call

```

SELECT *
FROM cfilter (
    ON (SELECT 1)
    PARTITION BY 1
    PASSWORD('beehive')
    INPUTTABLE('cfilter_test')
    OUTPUTTABLE('cfilter_test2')
    INPUTCOLUMNS('item')
    JOINCOLUMNS('tranid')
    OTHERCOLUMNS('storeid')
    DROPTABLE('true')
)
;

SELECT * FROM cfilter_test2 ORDER BY score DESC;

```

Example Output from Collaborative Filter

Table 10 - 273: Example output table

storeid	col1_item1	col1_item2	cntb	cnt1	cnt2	score	...
3	butter	flour	1	1	1	1	...
3	eggs	flour	1	1	1	1	...
3	flour	butter	1	1	1	1	...
3	flour	eggs	1	1	1	1	...

Table 10 - 273: Example output table (continued)

storeid	col1_item1	col1_item2	cntb	cnt1	cnt2	score	...
3	butter	eggs	1	1	1	1	...
3	eggs	butter	1	1	1	1	...
1	milk	eggs	3	4	3	0.75	...
1	butter	milk	3	3	4	0.75	...
1	milk	butter	3	4	3	0.75	...
1	eggs	milk	3	3	4	0.75	...
2	flour	eggs	2	2	3	0.6666666666666667	...
2	eggs	flour	2	3	2	0.6666666666666667	...
1	eggs	flour	2	3	2	0.6666666666666667	...
1	flour	eggs	2	2	3	0.6666666666666667	...
1	milk	flour	2	4	2	0.5	...
1	flour	milk	2	2	4	0.5	...
2	flour	butter	1	2	1	0.5	...
2	butter	flour	1	1	2	0.5	...
1	butter	eggs	2	3	3	0.444444444444444	...
1	eggs	butter	2	3	3	0.444444444444444	...
2	milk	eggs	1	1	3	0.333333333333333	...
2	eggs	butter	1	3	1	0.333333333333333	...
2	eggs	milk	1	3	1	0.333333333333333	...
2	butter	eggs	1	1	3	0.333333333333333	...
1	butter	flour	1	3	2	0.1666666666666667	...
1	flour	butter	1	2	3	0.1666666666666667	...

Error Messages

Table 10 - 274: Error Messages

Error Message	Reason
Function “cfilter” does not exist.	The cfILTER.zip SQLMR function is not installed.
Relation “<name of your output table>” already exists.	The output table already exists and the dropTable('true') argument was not used.
SQL-MR function CFILTER failed: Column in the OTHERCOLUMNS clause should be a column of the inputtable. found: “<one column that othercolumns passes in>”	A column that the othercolumns clause passes in is not a column of the input table.

Table 10 - 274: Error Messages (continued)

Error Message	Reason
SQL-MR function CFILTER failed: Column in the JOINCOLUMNS clause should be a column of the inputtable. found: “<one column that joincolumns passes in>”	A column that joincolumns clause passes in is not a column of the input table.
SQL-MR function CFILTER failed: Column in the INPUTCOLUMNS clause should be a column of the inputtable. found: “<one column that inputcolumns passes in>”	A column that the inputcolumns clause passes in is not a column of the input table.

CHAPTER 11 Graph Analysis

- Overview of Graph Functions
- nTree
- Single Source Shortest Path (SSSP)
- degrees (beta)
- triangle_finder (beta)
- rectangle_finder (beta)
- local_clustering_coefficient (beta)
- pagerank (beta)
- eigen_centrality (beta)



Tip: Before rerunning the examples in this chapter, manually remove the existing output tables.

Overview of Graph Functions

The Aster Database Analytics Foundation provides a set of functions that are widely used in social network analysis. These functions perform these tasks:

- Finding shortest paths
- Computing importance/influence scores
- Finding specific shapes in a graph

Graph Types

- Undirected Graph
- Directed Graph
- Multi-Graph

Undirected Graph

The Aster Database Analytics Foundation graph functions represent undirected graphs as tables of edge pairs in Aster Database, as shown in this example:

start_node	end_node
TED	ETHEL
ETHEL	TED
RICKY	FRED
FRED	RICKY
ETHEL	LUCY
LUCY	ETHEL
ETHEL	RANDY
RANDY	ETHEL
FRED	ETHEL
ETHEL	FRED
JOE	ETHEL
ETHEL	JOE
RANDY	RICKY
RICKY	RANDY
LUCY	FRED
FRED	LUCY

The functions represent each edge in the graph (a link between two nodes) as a pair of rows in opposite direction. For example, these two rows represent an edge between the TED and ETHEL nodes.

RANDY	RICKY
RICKY	RANDY

In the first row, the start node is TED and the end node is ETHEL. In the second row, the node order is reversed. Each edge in the graph must appear twice, but in the opposite direction.



Tip: Duplicate rows are not allowed in the graph table.

Directed Graph

A directed graph is represented in Aster Database as a table of edge rows (each edge is represented as one row in the table). The table has two columns (start_node and end_node).

start_node	end_node
ETHEL	TED
ETHEL	LUCY
ETHEL	RANDY
ETHEL	FRED
ETHEL	JOE
RANDY	RICKY
LUCY	FRED
FRED	RICKY

 **Tip:** Duplicate rows are not allowed in the graph table.

Multi-Graph

A multi-graph is represented in Aster Database as a table of edges, where each edge is represented by a row. However, unlike the undirected and directed graphs, there are no restrictions on the number of edge rows. For example, the table can have multiples of the same row.

start_node	end_node
ETHEL	TED
ETHEL	TED
ETHEL	TED
FRED	ETHEL
FRED	ETHEL
RANDY	RICKY
LUCY	FRED
FRED	LUCY

 **Tip:** In Aster Database, Undirected Graph \subset Directed Graph \subset Multi-Graph.

Multi-Graph with Duplicate Edges

In Aster Analytics 5.0-Release 2, the graph functions do not support a 'multi-graph' which has duplicate edges in one graph. As an example, consider a situation where people call one another multiple times in a telco call detail record (CDR).

In order to get the correct output of triangles and rectangles counts, etc. for a multi-graph with duplicate edges, you need to do a 'SELECT DISTINCT' on the CDR data before running the functions.

List of Graph Functions

Table 11 - 275: Aster Database Analytics Foundation Graph Functions

Category	Functions
Hierarchical Analysis	nTree
Path Analysis	Single Source Shortest Path (SSSP) (multi-graphs, directed graphs, undirected graphs)
Centrality (Influence scores)	degrees (beta) (directed graph, undirected graphs) local_clustering_coefficient (beta) (undirected graphs only) pagerank (beta) (directed graphs, undirected graphs) eigen_centrality (beta) (directed graphs, undirected graphs)
Shape Finder	triangle_finder (beta) (undirected graphs only) rectangle_finder (beta) (undirected graphs only)

nTree

Summary

nTree is a hierarchical analysis SQL-MR function which can build and traverse through tree structures on all worker machines. nTree reads the data only once from the disk and creates the trees in-memory. Note that this function requires the data to be partitionable and that each partition can fit in memory.

Most of the use-cases we have seen so far can be fit in memory. Each partition can consist of multiple trees. There is no restriction on the size of the tree. Since the data is not always clean and may contain cycles, there are different ways of handling the cycles.



Background

The following sections walk through various examples of use cases for nTree:

Equity Trading

A large stock buy or sell order is typically broken into a number of child orders in order to fulfill the trade with a number of counter parties. Each child order can be further broken into child orders. All transactions are stored in a single stock transaction table, with the parent order linked to its child orders by means of a parent_id.

A “root” order to sell, say, AAPL could result in a cascade of other AAPL transactions, all of which can trace their ancestry back to the original order. For example, you could have order_id 1 to sell 100 of AAPL lead to order 2 and 3 for 70 and 30 shares each. Those, in turn, could be further split up. Each row of this transaction table would include that row’s order_id as well as its parent_id.

The broker needs to be able to identify the root order for each transaction. With SQL this would require an unknown number of self-joins. With Aster Database SQL-MapReduce we can partition the data by ticker symbol and then by date, and use the nTree function to create a tree from each root order.

Social Networking

Social networks use multiple data sources to identify people and their relationships. For example, a user-user connection graph defines explicit connections the users have created on the network, a user-person invitation graph shows a mixture of user-user connections and user-email connections, and finally, address book data provides a user-email graph.

It is often important for a social network to clean up its data by detecting when a person has multiple accounts on the network. In a case like this, you can use nTree to generate a tree for every account. You can then compare these trees to find trees that are very likely to have the same person as the root-node.

Usage

Syntax (version 1.0)

```
SELECT * FROM NTREE
(
    ON { input_table | view | query }
    PARTITION BY partition_columns
    [ORDER BY ordering_columns]
    ROOT_NODE(expression)
    NODE_ID(expression)
    PARENT_ID(expression)
    MODE('UP' | 'DOWN')
    ALLOW_CYCLES('true' | 'false')
    STARTS_WITH(expression | 'root' | 'leaf')
    OUTPUT('END' | 'ALL')
    RESULT(aggregate(expression) as alias)
    [LOGGING ('true' | 'false')]
) ;
```

Arguments

Arguments used for creating the tree:

<i>ROOT_NODE</i>	Required	This is an SQL expression which returns a boolean value, which is used to define the root nodes of the trees. All the tuples which evaluate to true for this SQL-expression are considered root nodes.
<i>NODE_ID</i>	Required	Each row represents a node. The NODE_ID is the unique identifier for the node. This argument is an SQL expression which uniquely identifies a node in the dataset. Note that the same node can appear multiple times with different parents.
<i>PARENT_ID</i>	Required	This is an SQL expression which evaluates the value of the ID for the parent node.
<i>ALLOW_CYCLES</i>	Required	If the argument is set to 'true', cycles are allowed in the tree. If set to 'false', an exception is thrown when there is a cycle in the dataset. The default value is 'false'.

Arguments used for selecting the type of operation to be performed:

<i>MODE</i>	Required	This is the argument used to select the type of traversal from the STARTS_WITH nodes. It can be either 'UP' or 'DOWN'. If the value is 'UP', the function will start from the STARTS_WITH node and traverse up, towards the root node. If the value is 'DOWN', the function will start at the STARTS_WITH node and traverse down, towards the leaf nodes.
-------------	----------	---

Argument to identify the start node for the push-down operation:

<i>STARTS_WITH</i>	Required	This is an SQL-boolean Expression which is used to define the node from which to start the tree traversal.
--------------------	----------	--

Arguments used for managing the function's output:

<i>OUTPUT</i>	Required	This argument can take two values: 'ALL' or 'END'. 'ALL' outputs one result tuple at every node along the path traversal. 'END' outputs one result tuple when the traversal reaches the end of path. The default value is 'END'.
<i>LOGGING</i>	Required	('true' 'false'). If true, then log messages is printed.

RESULT

- Required This argument specifies which aggregate operations can be performed while traversing the tree. Note that when using an alias, it must not be the same as the name of a column in the input table, or it is ignored. The supported aggregate types are: PATH, SUM, LEVEL, MAX, MIN, IS_CYCLE, AVG, PROPAGATE, as described here:
- *PATH(expr)* outputs the path from the STARTS_WITH node to the last traversed node.
 - *SUM(expr)* outputs the sum of *expr* (expression) on all nodes from the STARTS_WITH node to the last traversed node.
 - *LEVEL(*)* outputs the number of hops from the STARTS_WITH node to the last traversed node. Note that this is the number of hops from the STARTS_WITH node and NOT the Root node.
 - *MAX(expr)* outputs the maximum value encountered so far from the STARTS_WITH node to the last traversed node.
 - *MIN(expr)* outputs the minimum value encountered so far from the STARTS_WITH node to the last traversed node.
 - *IS_CYCLE(*)* outputs the cycle (if any).
 - *AVG(expr)* outputs the average value so far from the STARTS_WITH node to the last traversed node.
 - *PROPAGATE(expr)* evaluates the expression *expr* on the STARTS_WITH node and propagates it to all the nodes along the path.

Cycles in nTree

Cycles are unidirectional, meaning that the output when changing from MODE('UP') to MODE('DOWN') is different.

As an example, consider the employee_table described in [Table 11 - 276](#).

This query uses MODE('UP'):

```
SELECT * FROM ntree (ON emp_table_aster
    PARTITION BY department
    ORDER BY order_column
    ROOT_NODE(mgr_id = 'none')
    PARENT_ID(mgr_id)
    NODE_ID(id)
    STARTS_WITH(id = '11')
    MODE ('UP')
    OUTPUT ('END')
    RESULT(PATH(name) AS path, path(id) AS path2,is_cycle(*))
    ALLOW_CYCLES('true'))
    ORDER BY path, path2;
```

The output is:

id	path	path2	is_cycle
12	Test2->Test1->Test3	11->10->12	10
10	Test2->Test4->Test3->Test1	11->15->12->10	12
14	Test2->Test4->Test8	11->15->14	11
10	Test2->Test4->Test8->Test3->Test1	11->15->14->12->10	12
(4 rows)			

The same query using MODE('DOWN') generates this output:

id	path	path2	is_cycle
13	Test2->Test7	11->13	
15	Test2->Test8->Test4	11->14->15	11
18	Test2->Test8->Test4->Test5->Test9->Test6	11->14->15->16->17->18	16

(3 rows)

As these examples show, a cycle in MODE('UP') may not be seen in MODE('DOWN'). This is because cycles are considered to be unidirectional in nTree.

Examples

Example 1: Find an employee's chain of managers

This example uses nTree to find an employee's chain of managers all the way up the reporting hierarchy. The manager chain is expressed in the output as a path from the direct manager all the way up to the top level manager. Note that specifying OUTPUT ('END') only outputs one tuple of data for the last row in the path.

Example 1 Input Data to nTree

Table 11 - 276: Example input table: employee_table

emp_id	emp_name	mgr_id	salary
100	Don	null	10k
200	Pat	100	8k
300	Donna	100	8k
400	Kim	200	6k
500	Fred	400	4k

Example 1 nTree SQL-MR Call

Here, we start with employee 400 and follow the graph UP, as specified by the *MODE* argument:

```
select *
FROM ntree
(ON employee_table
  PARTITION BY 1
  ROOT_NODE(mgr_id is null)
  NODE_ID(emp_id)
  PARENT_ID(mgr_id)
  MODE('UP')
  ALLOW_CYCLES('true')
  STARTS_WITH(emp_id=400)
  OUTPUT('END')
  RESULT(path(emp_name) as path)
) ;
```

Example 1 Output

Table 11 - 277: Example Output from nTree

id	path
500	Kim->Pat->Don

Example 2: Identify chains of managers by name and id

In this example, we want to identify each chain of managers in the dataset. The manager chain will again be expressed in the output as a path, only this time the function will output the path by name and by id. Note that specifying `OUTPUT ('ALL')` outputs all tuples of data. Also, the data is partitioned by department in this example.

Example 2 Input Data

Table 11 - 278: Example input table: emp_table_aster (partitioned on department)

department	order_column	id	name	salary	mgr_id
aster	1	7	Don	20	2
aster	2	2	Pat	30	3
aster	3	3	Donna	60	6
aster	4	9	Kim	50	5
aster	5	4	Fred	40	4
aster	6	5	Mark	70	7
aster	7	6	Rob	10	1
aster	8	5	Mark	10	1
aster	9	1	Dave	10	none
aster	10	1	Dave	10	9
teradata	1	10	Test1	10	12
teradata	2	11	Test2	20	10
teradata	3	12	Test3	30	10
teradata	4	15	Test4	40	12
teradata	5	16	Test5	50	12
teradata	6	18	Test6	60	17
teradata	7	11	Test2	20	15
teradata	8	13	Test7	70	11
teradata	9	14	Test8	80	11
teradata	10	14	Test8	80	12

Table 11 - 278: Example input table: emp_table_aster (partitioned on department) (continued)

department	order_column	id	name	salary	mgr_id
teradata	11	15	Test4	40	14
teradata	12	16	Test5	50	15
teradata	13	17	Test9	90	16
teradata	14	16	Test5	50	18
teradata	1	10	Test1	10	none

Example 2 SQL-MR Call

```
SELECT *
FROM ntree
( ON emp_table_aster
  PARTITION BY department
  ORDER BY order_column
  ROOT_NODE(mgr_id = 'none')
  PARENT_ID(mgr_id)
  NODE_ID(id)
  STARTS_WITH('ROOT')
  MODE('DOWN')
  OUTPUT('ALL')
  RESULT(PATH(name) as path,
         path(id) as path2)
  ALLOW_CYCLES('true')
) order by path, path2;
```

Example 2 Output

Table 11 - 279: Example output table

id	path	path2
1	Dave	1
5	Dave->Mark	1->5
9	Dave->Mark->Kim	1->5->9
6	Dave->Rob	1->6
3	Dave->Rob->Donna	1->6->3
2	Dave->Rob->Donna->Pat	1->6->3->2
7	Dave->Rob->Donna->Pat->Don	1->6->3->2->7
5	Dave->Rob->Donna->Pat->Don->Mark	1->6->3->2->7->5
9	Dave->Rob->Donna->Pat->Don->Mark->Kim	1->6->3->2->7->5->9
10	Test1	10
11	Test1->Test2	10->11
13	Test1->Test2->Test7	10->11->13
14	Test1->Test2->Test8	10->11->14

Table 11 - 279: Example output table (continued)

id	path	path2
15	Test1->Test2->Test8->Test4	10->11->14->15
16	Test1->Test2->Test8->Test4->Test5	10->11->14->15->16
17	Test1->Test2->Test8->Test4->Test5->Test9	10->11->14->15->16->17
18	Test1->Test2->Test8->Test4->Test5->Test9->Test6	10->11->14->15->16->17->18
12	Test1->Test3	10->12
15	Test1->Test3->Test4	10->12->15
11	Test1->Test3->Test4->Test2	10->12->15->11
13	Test1->Test3->Test4->Test2->Test7	10->12->15->11->13
14	Test1->Test3->Test4->Test2->Test8	10->12->15->11->14
16	Test1->Test3->Test4->Test5	10->12->15->16
17	Test1->Test3->Test4->Test5->Test9	10->12->15->16->17
18	Test1->Test3->Test4->Test5->Test9->Test6	10->12->15->16->17->18
16	Test1->Test3->Test5	10->12->16
17	Test1->Test3->Test5->Test9	10->12->16->17
18	Test1->Test3->Test5->Test9->Test6	10->12->16->17->18
14	Test1->Test3->Test8	10->12->14
15	Test1->Test3->Test8->Test4	10->12->14->15
11	Test1->Test3->Test8->Test4->Test2	10->12->14->15->11
13	Test1->Test3->Test8->Test4->Test2->Test7	10->12->14->15->11->13
16	Test1->Test3->Test8->Test4->Test5	10->12->14->15->16
17	Test1->Test3->Test8->Test4->Test5->Test9	10->12->14->15->16->17
18	Test1->Test3->Test8->Test4->Test5->Test9->Test6	10->12->14->15->16->17->18

Support for Very Deep Trees with nTree

When working with very deep trees, nTree may throw a `java.lang.StackOverflowError`. This is because by default, the Java stack size in many systems supports trees of roughly up to 7500 levels deep. When working with trees that have more than 7500 levels, it may be necessary to increase the Java stack size.

To increase the Java stack size, add the following line in `/etc/profile`:

```
export JAVA_TOOL_OPTIONS=' -Xss24M'
```

This must be done on all worker nodes, but is not necessary on the coordinator (queen) node.

Using nTree on a pre-5.0 Aster Database

The nTree function was developed for Aster Database 5.0 and involves some changes to the database planner. nTree is a pre-packaged SQL-MR function, which means that the function does not need to be installed separately. This function does not come pre-installed on Aster Database versions earlier than 5.0.

If you need to use nTree on a older cluster (pre-5.0) you can install and use a more verbose version of nTree. In this case, you would need to install the nTree SQL-MR function on the cluster using the \install command. nTree is part of the “4.6.3 Analytics Function Preview”, which can be obtained from Teradata Aster Support.

In this more verbose version of the nTree SQL-MR function (pre-5.0) you will need to pull up all the expressions used in the ROOT_NODE, NODE_ID, PARENT_ID, STARTS_WITH, and RESULT clauses to the SELECT list of the input table, and reference them using the input column index in the argument clauses.

Example of pre-5.0 nTree

The more verbose version of the sample query in example 2 would look like this:

```
SELECT * FROM nTree(
    ON (SELECT mgr_id = 'none', mgr_id, id, name, department
        from emp_table)
    PARTITION BY department
    ROOT_NODE('0')
    NODE_ID('2')
    PARENT_ID('1')
    STARTS_WITH('ROOT')
    MODE ('DOWN')
    RESULT('--function', 'path', '-oncolumn', '3', 'outputColumnName',
          'path', '-endoutput', '-function', 'path', '-oncolumn', '2',
          '-outputColumnName', 'path2', '-endoutput')
    OUTPUT('ALL')
    ALLOW_CYCLES('true')
);
```

All the expressions are now part of the SELECT clause of the input table. These columns are then referenced in the argument clauses using the column indexes. For example, expression in the ROOT_NODE ('mgr_id=null') clause has been moved to the select clause and the column index (0) has been used in the ROOT_NODE clause. It is as simple as this for the ROOT_NODE, NODE_ID, PARENT_ID, and STARTS_WITH clauses.

For the RESULT clause, the aggregates which need to be performed must be specified in a more verbose way. The name of the aggregate is specified after the '-function' keyword. The column index of the input column is specified after the '-oncolumn' keyword and the name of the output column is specified after the '-ouputColumnName' keyword. In addition, each aggregate should end with a '-endoutput' keyword.

Single Source Shortest Path (SSSP)

Summary

Given a graph with vertices and edges between these vertices, the Single-Source Shortest Path (SSSP) function finds the shortest paths from a given vertex to all the other vertices in the graph.



Installation

Before you can use the SSSP function, you must install it. To do so, copy the `ssspDriver.jar` file, which you can find in the `Analytics_Foundation.zip` package, to the system from which you plan to run the SSSP function.

For example, if you plan to use the function on the queen, copy the `ssspDriver.jar` file to the Queen. If you plan to call SSSP from a client node, copy the `ssspDriver.jar` file to the client node.

For more information about installing analytics functions, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.0)

Usage on the queen

For clarity, we break the command line below at each argument flag. Each value shown in *italics* is one that you should replace with the right value your installation:

```
$ java
-classpath
path_1/ssspDriver.jar:
path_2/ncluster-sqlmr-api.jar:
path_3/noarch-ncluster-jdbc-driver.jar
com.asterdata.sqlmr.analytics.path_analysis.sssp.ssspDriver

-domain=ip_address
-database=database_name
-userid=user_id
-password=password

-inputTable=input_table_name
-sourceColumnName=source_column_name
-destinationColumnName=destination_column_name
-startNode=start_node_number

-outputTable=output_table_name
```

Usage on a remote (non-queen) machine

For clarity, we break the command line below at each argument flag. Each value shown in *italics* is one that you should replace with the right value your installation:

```
user@machine:~$ java
    -classpath ssspDriver.jar:path_for_JDBC_and_nculster-sqlmr jar files
    com.asterdata.sqlmr.analytics.path_analysis.sssp.ssspDriver
    -domain=host:port
    -database=database_name
    -userid=user_id
    -password=password
    -inputtable=input_table_name
    -outputtable=output_table_name
    -source=source_column
    -destination=destination_column
    -startnode=start_node
```

Arguments

The format of the command-line arguments follows the usual Java style. That is, you pass each setting as a pair in the form *-argument_name=argument_value*, and a space separates each setting from the next. The command-line arguments used to invoke this function are:

<i>DOMAIN</i>	Optional	Has the form, <i>host:port</i> . The <i>host</i> is the Aster Database queen's IP address or hostname. To specify an IPv6 address, enclose the host argument in square brackets, for example: [::1]:2406. The <i>port</i> is the port number that the queen is listening on. The default is the Aster standard port number (2406). For example: DOMAIN(10.51.23.100:2406)
<i>DATABASE</i>	Required	This is the name of the database where the input table is present. For example: -database=beehive
<i>USERID</i>	Required	The Aster Database user name of the user. For example: -userid=beehive
<i>PASSWORD</i>	Required	The Aster Database password of the user. For example: -password=beehive
<i>INPUTTABLE</i>	Required	The name of the input table. This is the table containing the list of edges. The input table should have at least two columns: the <i>source</i> column and the <i>destination</i> column.
<i>OUTPUTTABLE</i>	Required	This is the name of the output table where you wish to save the results. The output table will contain the columns listed in the section "Output," below.
<i>SOURCE</i>	Required	The name of the input table column that contains the source vertex.
<i>DESTINATION</i>	Required	The name of the input table column that contains the destination vertex.
<i>STARTNODE</i>	Required	The node from which the shortest path to all the other vertices should be computed.

Input

Input table containing the list of edges. Input table should contain at-least two columns, *source* column and the *destination* column

Output

You use the `-outputtable` argument to specify the destination table for the function's results.
IMPORTANT! Please note that if a table with this name exists already, that table is dropped.

The output table will contain the following columns:

<i>node</i>	Represents a vertex in the graph.
<i>seen</i>	True/false value indicating whether this vertex is reachable from the start vertex.
<i>points_to</i>	List of all the reachable vertices from this vertex.
<i>distance_from_start</i>	Distance from the start node to this vertex. If this vertex is <i>not</i> reachable from the start node, then this value is -1.
<i>path_from_start</i>	The shortest path from the start node to this node. If this vertex is not reachable from the start node, this value is empty.

Example

Input

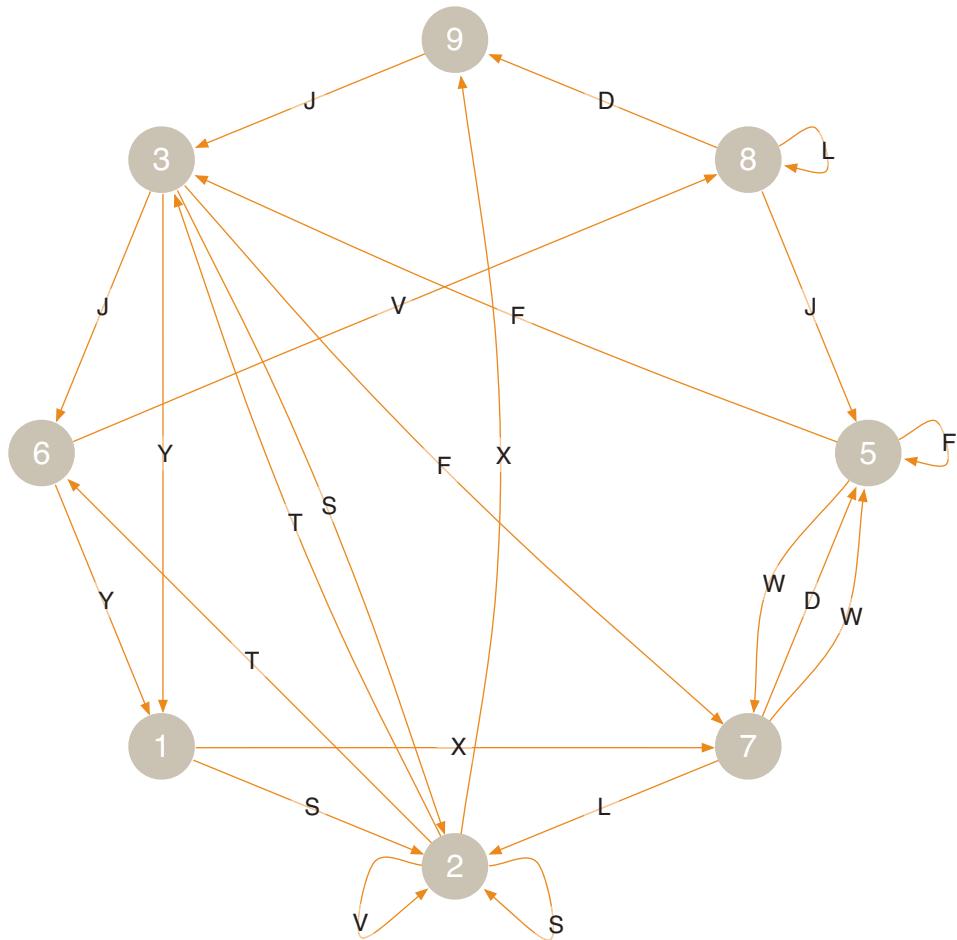
Table 11 - 280: Example input table: graph

source	name	destination
1	"san carlos"	2
3	"san jose"	6
5	"san francisco"	3
6	"new york"	1
7	"dallas"	5
7	"LA"	2
2	"seattle"	6
6	"las vegas"	8
2	"Mexico"	9
7	"washington"	5
2	"san carlos"	2
8	"san jose"	5
3	"san francisco"	7
3	"new york"	1

Table 11 - 280: Example input table: graph (continued)

source	name	destination
8	"dallas"	9
8	"LA"	8
2	"seattle"	3
2	"las vegas"	2
1	"Mexico"	7
5	"washington"	7
3	"san carlos"	2
9	"san jose"	3
5	"san francisco"	5

This graph can be represented by this diagram.



Sample Usage

```
user@machine:~$ java
  -jar ssspDriver.jar
  -startNode=2
  -domain=10.51.23.100
  -database=beehive
  -inputTable=graph
  -outputTable=shortestDistance
  -sourceColumnName=source
  -destinationColumnName=destination
  -userID=beehive
  -password=beehive
```

The first line invokes Java. Lines two through five include the appropriate Java libraries to run the driver. Line six is the name of the Java class we want to run. All lines afterward are program arguments.

Example Output Data from SSSP

Table 11 - 281: Example output table

Node	Seen	Points_To	Distance_from_start	Path_from_start
2	t	2,3,6,9	0	empty
6	t	1,8	1	6
8	t	5,8,9	2	6,8
1	t	2,7	2	6,1
3	t	1,2,6,7	1	3
5	t	3,5,7	3	3,7,5
7	t	2,5	2	3,7
9	t	3	1	9

Error Messages

In response to user error, the SSSP operator may print the following error messages:

ERROR: Please provide all of the following arguments: -domain -database -inputTable -outputTable -userid -password -sourceColumnName -destinationColumnName -startNode.

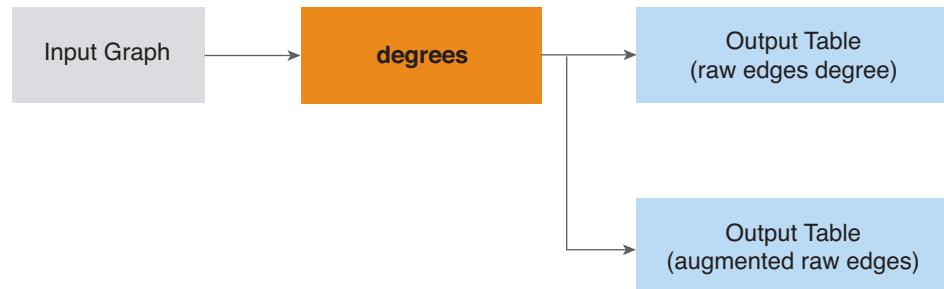
REASON: One or more of the following arguments are missing. Arguments can be any of the following: -domain -database -table -userid -password -sourceColumnName -destinationColumnName -startNode.

REASON: One or more of the arguments provided is not valid.

degrees (beta)

Summary

This function generates the in-degree and out-degree tables for a directed graph. For an undirected graph, it just generates one degree table. This function can also generate the augmented edges table, which you can use in other graph functions, such as [triangle_finder \(beta\)](#) and [rectangle_finder \(beta\)](#).



Background

The degrees function is the simplest way to calculate the centrality measure of a graph. This function is just a wrapper function that issues several SQL queries to generate the degree tables.

Usage

Syntax (version 1.0)

```
SELECT * FROM DEGREES (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
    [ DIRECTED('true | t | false | f') ]
    [ AUGMENTED('true | t | false | f') ]
) ;
```

Arguments

DOMAIN	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[:: 1]:2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
DATABASE	Optional	The name of the database where the input table resides. The default database is beehive.

<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default is beehive.
<i>PASSWORD</i>	Required	The Aster Database user password.
<i>INPUTTABLE</i>	Required	The raw edges table.
<i>STARTNODE</i>	Optional	The name of the start node column. If not specified, the first column is picked.
<i>ENDNODE</i>	Optional	The name of the end node column. If not specified, the second column is picked.
<i>DIRECTED</i>	Optional	Specifies whether the input graph is directed or un-directed. The default value is <code>t</code> (true).
<i>AUGMENTED</i>	Optional	Specifies whether the function generates an augmented edges table. The default value is <code>t</code> (true).

Example

Table 11 - 282: Example input table: raw_edges_string

vertex1	vertex2
TED	ETHEL
ETHEL	TED
RICKY	FRED
FRED	RICKY
ETHEL	LUCY
LUCY	ETHEL
ETHEL	RANDY
RANDY	ETHEL
FRED	ETHEL
ETHEL	FRED
JOE	ETHEL
ETHEL	JOE
RANDY	RICKY
RICKY	RANDY
LUCY	FRED
FRED	LUCY

Sample SQL-MR Call

```
SELECT * FROM DEGREES (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    inputTable('raw_edges_string')
    directed('f')
    augmented('t')
) ;
```

Output

This function displays these messages:

Table 11 - 283: Console Messages

message
Successful.
This is an undirected graph.
Table 'raw_edges_string_degree' created.
Table 'augmented_raw_edges_string' created.

To display the raw_edges_string_degree output table, run this query:

```
SELECT * FROM raw_edges_string_degree ORDER BY degree DESC;
```

Table 11 - 284: Example output table: raw_edges_string_degree

vertex1	degree
ETHEL	5
FRED	3
RICKY	2
LUCY	2
RANDY	2
TED	1
JOE	1

To display the augmented_raw_edges_string output table, run this query:

```
SELECT * FROM augmented_raw_edges_string;
```

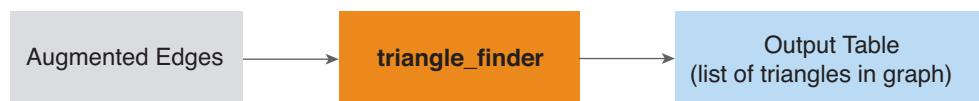
Table 11 - 285: Example output table: augmented_raw_edges_string

vertex1	vertex2	outdegree	indegree
TED	ETHEL	1	5
ETHEL	TED	5	1
RICKY	FRED	2	3
FRED	RICKY	3	2
ETHEL	LUCY	5	2
LUCY	ETHEL	2	5
ETHEL	RANDY	5	2
RANDY	ETHEL	2	5
FRED	ETHEL	3	5
ETHEL	FRED	5	3
JOE	ETHEL	1	5
ETHEL	JOE	5	1
RANDY	RICKY	2	2
RICKY	RANDY	2	2
LUCY	FRED	2	3
FRED	LUCY	3	2

triangle_finder (beta)

Summary

This function finds triangles in an undirected graph. It is a driver function that calls the triangleFinderMap and triangleFinderReduce functions. The triangle_finder function generates a table listing the triangles in the graph.



Background

Triangles, or 3-cycles, can be the basis for analyzing graphs and subgraphs. In particular, enumerating triangles constitutes most of the work in identifying the dense subgraphs called trusses. MapReduce offers a good framework for locating triangles.

On the other hand, triangle-free graphs are of particular interest to some researchers. This algorithm also provide a way to find whether a graph is triangle-free or not.

Usage

Syntax (version 1.0)

```
SELECT * FROM triangle_finder (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    [ STARTNODE('vertex1') ]
    [ ENDNODE('vertex2') ]
    [ OUTDEGREE('degree1') ]
    [ INDEGREE('degree2') ]
    [ INVOKEMETHOD('single | multiple') ]
) ;
```

Arguments

DOMAIN	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[::1]:2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
DATABASE	Optional	The name of the database where the input table resides. The default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default value is beehive.
PASSWORD	Required	The Aster Database user password.
INPUTTABLE	Required	The name of the input table describing the graph. This table should be an augmented edges table. This table can be generated by the degrees (beta) function.
OUTPUTTABLE	Required	The name of the output table. If it already exists, an error is thrown.
STARTNODE	Optional	The column name of the start node in the input table. If not specified, the first column is picked.
ENDNODE	Optional	The column name of the end node in the input table. If not specified, the second column is picked.
OUTDEGREE	Optional	The name of the out degree (number of edges coming out from the startnode) column. If not specified, the third column is picked.

INDEGREE	Optional	The name of the in degree (number of edges going into the endnode) column. If not specified, the fourth column is picked.
INVOKEMETHOD	Optional	This function can be run on both old and new platforms. For 4.6.x and earlier versions, use 'single'; for 5.0 and later versions, use 'multiple'. Default value is 'multiple'.



Tip: This function represents each edge of an undirected graph using two rows in the database.

Example

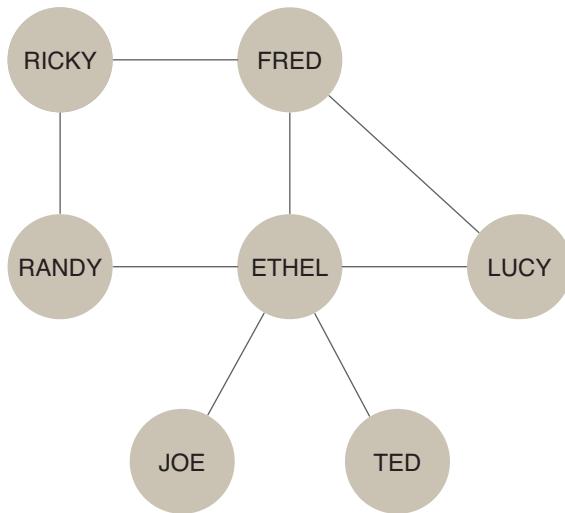
Sample Input

Table 11 - 286: Example input table: augmented_edges (can be generated by the degrees function)

vertex1	vertex2	degree1	degree2
FRED	ETHEL	3	5
ETHEL	FRED	5	3
ETHEL	LUCY	5	2
LUCY	ETHEL	2	5
LUCY	FRED	2	3
FRED	LUCY	3	2
JOE	ETHEL	1	5
ETHEL	JOE	5	1
TED	ETHEL	1	5
ETHEL	TED	5	1
RICKY	FRED	2	3
FRED	RICKY	3	2
ETHEL	RANDY	5	2
RANDY	ETHEL	2	5
RANDY	RICKY	2	2
RICKY	RANDY	2	2

[Figure 11](#) is a graphical representation of the graph described by the input table.

Figure 11: Input Graph



Sample SQL-MR Call

```
SELECT * FROM triangle_finder (
    ON (SELECT 1)
    PARTITION BY 1

    database('beehive')
    userid('beehive')
    password('beehive')

    inputTable('augmented_edges')
    outputTable('triangles')
) ;
```

Sample output

The triangle_finder function displays these messages:

Table 11 - 287: triangle_finder Console Messages

message
Successful.
Table 'triangles' created.

To display the triangles discovered by the function, run this SQL query:

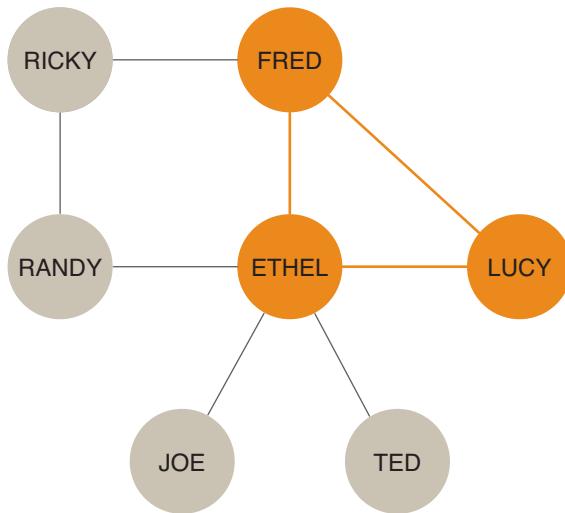
```
SELECT * FROM triangles;
```

Table 11 - 288: triangle_finder Output

node1	node2	node3
LUCY	ETHEL	FRED

Figure 12 is a graphical representation of the results of this function.

Figure 12: Triangles in the Graph

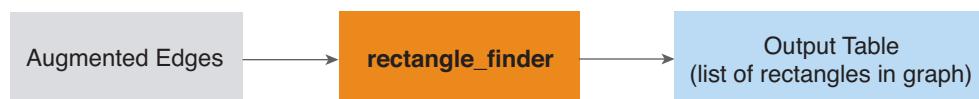


Tip: This function does not generate graphical representation of the results. This graphic was generated manually.

rectangle_finder (beta)

Summary

This function finds rectangles in an undirected graph.



Background

The job of enumerating rectangles (4-cycles) is similar to that of enumerating triangles. For more information, see “triangle_finder (beta)” on page 444.

Usage

Syntax (version 1.0)

```
SELECT * FROM rectangle_finder (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')
    [ STARTNODE('vertex1') ]
    [ ENDNODE('vertex2') ]
    [ OUTDEGREE('degree1') ]
    [ INDEGREE('degree2') ]
) ;
```

Arguments

DOMAIN	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[::1]:2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
DATABASE	Optional	The name of the database where the input table resides. The default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default value is beehive.
PASSWORD	Required	The Aster Database user password.
INPUTTABLE	Required	The input table describing the input graph. This table should be an augmented edges table. This table can be generated by the degrees (beta) function.
OUTPUTTABLE	Required	Name of the output table. If it already exists, an error is thrown.
STARTNODE	Optional	The column name of the start node in the input table. If not specified, the first column is picked.
ENDNODE	Optional	The column name of the end node in the input table. If not specified, the second column is picked.
OUTDEGREE	Optional	The name of the out degree (number of edges coming out from the startnode) column. If not specified, the third column is picked.
INDEGREE	Optional	The name of the in degree (number of edges going into the endnode) column. If not specified, the fourth column is picked.

Example

Sample Input

Table 11 - 289: Example input table: augmented_edges (can be generated by the degrees function)

vertex1	vertex2	degree1	degree2
TED	ETHEL	1	5
ETHEL	TED	5	1
RICKY	FRED	2	3
FRED	RICKY	3	2
ETHEL	LUCY	5	2
LUCY	ETHEL	2	5
ETHEL	RANDY	5	2
RANDY	ETHEL	2	5
FRED	ETHEL	3	5
ETHEL	FRED	5	3
JOE	ETHEL	1	5
ETHEL	JOE	5	1
RANDY	RICKY	2	2
RICKY	RANDY	2	2
LUCY	FRED	2	3
FRED	LUCY	3	2

[Figure 11](#) is a graphical representation of the graph described by the input table.

Sample SQL-MR Call

```
SELECT * FROM rectangle_finder (
    ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('augmented_edges')
    outputTable('rectangles')
) ;
```

Sample output

The rectangle_finder function displays these messages:

Table 11 - 290: Console Messages

message
Successful.
Table 'rectangles' created.

To display the found rectangles, run this SQL query:

```
SELECT * FROM rectangles;
```

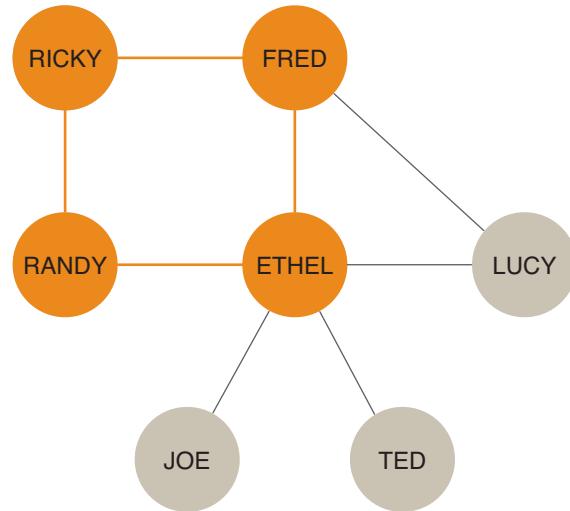
Table 11 - 291 displays the results.

Table 11 - 291: Example output table: rectangle_finder

node1	node2	node3	node4
FRED	ETHEL	RANDY	RICKY

Figure 13 is a graphical representation of the results of this function.

Figure 13: Rectangles in the Graph

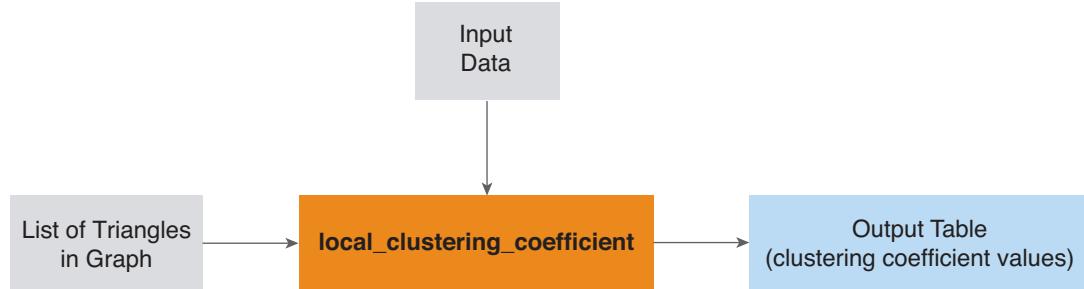


Tip: This function does not generate graphical representation of the results. This graphic was generated manually.

local_clustering_coefficient (beta)

Summary

This function computes the local clustering coefficient for nodes whose local clustering coefficient is not zero in an undirected graph.



Background

The local clustering coefficient of a vertex in a graph quantifies how close its neighbors are to being a clique (complete graph). Clustering coefficients determine whether a graph is a small-world network.

To compute the clustering coefficient for a given node (source node), this function uses this formula:

$$\text{local clustering coefficient for a node} = \frac{2 * (\text{number of triangles})}{n * (n-1)}$$

The number of triangles is the number of direct connections between the node's neighbors and n is the number of edges of the node.

This is effectively the triangle_finder algorithm and is central to many applications in graph analysis.

Usage

Syntax (version 1.0)

```

SELECT * FROM local_clustering_coefficient (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    TRIANGLES('triangles_table')
    OUTPUTTABLE('output_table')
    [ NODECOL('node_column_name') ]
    [ DEGREECOL('degree_column_name') ]
)
;
```

Arguments

<i>DOMAIN</i>	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, [:: 1]:2406). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, DOMAIN(10.51.23.100:2406).
<i>DATABASE</i>	Optional	The name of the database where the input table resides. The default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user running this function. The default value is beehive.
<i>PASSWORD</i>	Required	The Aster Database user password.
<i>INPUTTABLE</i>	Required	Name of the degrees table. To dramatically reduce the amount of computation needed to calculate the local clustering coefficient values, eliminate any vertex with a degree of 1 because the vertex can never be in a triangle, and its local clustering coefficient is always 0.
<i>TRIANGLES</i>	Required	Name of the triangles table, which should be the result of the triangle_finder (beta) function.
<i>OUTPUTTABLE</i>	Required	Name of the output table. If it already exists, an error is thrown.
<i>NODECOL</i>	Optional	Node column name in the input table. If not specified, the first column is picked.
<i>DEGREECOL</i>	Optional	Degree column name in the input table. If not specified, the second column is picked.

Example

Sample Input

Table 11 - 292: Example input table: degrees

node	degree
TED	1
RICKY	2
ETHEL	5
FRED	3
JOE	1
RANDY	2
LUCY	2

Table 11 - 293 lists the contents of the second input table, triangles. This table is generated by the [triangle_finder \(beta\)](#) function.

Table 11 - 293: Second input table: triangles

node1	node2	node3
LUCY	FRED	ETHEL

Sample SQL-MR Call

```
SELECT * FROM local_clustering_coefficient (
    ON (SELECT 1)
    PARTITION BY 1
    database('beehive')
    userid('beehive')
    password('beehive')
    inputTable('degrees')
    triangles('triangles')
    outputTable('local_cc')
    nodeCol('node')
    degreeCol('degree')
) ;
```

Sample Output

The local_clustering_coefficient function displays these messages:

Table 11 - 294: Console Messages

message
Successful.
Table 'local_cc' created.

To display the output table, run this query:

```
SELECT * FROM local_cc;
```

Table 11 - 295: Local Clustering Coefficient Values

vertex	local_cc
ETHEL	0.10000000000000000000000000000000
FRED	0.16666666666666666667
LUCY	1.0000000000000000000000000000000

pagerank (beta)

Summary

This function computes PageRank for a directed graph.



Background

PageRank is a link analysis algorithm. It assigns a numerical weighting (between 0 and 1) to each node in a graph, with the purpose of *measuring* its relative importance within the graph. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The sum of PageRanks is 1.

Usage

Syntax (version 1.0)

```
SELECT * FROM PAGERANK (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')
    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')

    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
    [ DAMPFACTOR('damping_factor') ]
    [ MAXITERNUM('max_iter') ]
    [ THRESHOLD('threshold') ]
    [ DELIMITER('delimiter') ]
```

Arguments

DOMAIN	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[:: 1] :2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
DATABASE	Optional	The name of the database where the input table resides. The default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default value is beehive.
PASSWORD	Required	The Aster Database user password.
INPUTTABLE	Required	The table describing the graph.
OUTPUTTABLE	Required	The name of the output table. If it already exists, an error is thrown.
STARTNODE	Optional	The name of the start node column. If not specified, the first column is picked.
ENDNODE	Optional	The name of the end node column. If not specified, the second column is picked.
DAMPFACTOR	Optional	A value between 0 and 1 that is used in the pagerank formula. If not specified, then 0.85 is used.
MAXITERNUM	Optional	The maximum iteration number the algorithm goes through. If not specified, 20 is used.
THRESHOLD	Optional	A value that specifies the algorithm convergence criteria. If not specified, 0.0001 is used.
DELIMITER	Optional	Allowed values are the space character (' '), '#', '\$', '%', '&'. The default is '' (space). This delimiter is used to concatenate nodes as a string. Any node should not have a delimiter as one of its characters.

Example

Sample Input

Table 11 - 296: Example input table: raw_edges

vertex1	vertex2
TED	ETHEL
ETHEL	TED
RICKY	FRED
FRED	RICKY
ETHEL	LUCY
LUCY	ETHEL
ETHEL	RANDY
RANDY	ETHEL
FRED	ETHEL
ETHEL	FRED
JOE	ETHEL
ETHEL	JOE
RANDY	RICKY
RICKY	RANDY
LUCY	FRED
FRED	LUCY

[Figure 11](#) is a graphical representation of the graph described by the input table.

Sample SQL-MR Call

```
SELECT * FROM PAGERANK (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    inputTable('raw_edges')
    outputTable('pagerank_string')
) ;
```

Sample Output

This function displays these messages:

Table 11 - 297: Console Messages

message
Total non-null nodes number: 7.
Non-null start nodes number: 7.
sq_err
0.104688775510204
0.0486850727040816
0.021147312126116
0.0108719445735185
0.00556565993396361
0.00299959444999363
0.00162083792424099
8.89524852625799E-4
4.89020518819743E-4
2.70169020450999E-4
1.4939934101316E-4
8.27564015258847E-5

Successful. Algorithm converged.

Total iteration number: 12

The result is stored in table 'pagerank_string'.

To display the output table, run this query:

```
SELECT * FROM pagerank_string ORDER BY pagerank DESC;
```

Table 11 - 298: Node Pagerank Values

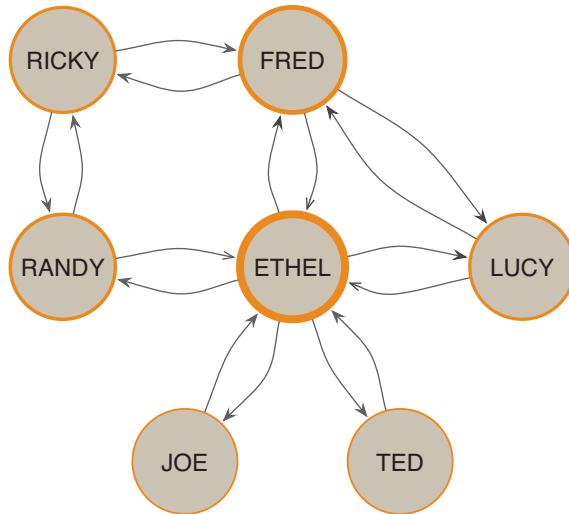
node	pagerank
ETHEL	0.298492305386488
FRED	0.179692023732142
RANDY	0.127438663588184

Table 11 - 298: Node Pagerank Values (continued)

node	pagerank
RICKY	0.124249195382751
LUCY	0.123415758580048
JOE	0.0733560266651933
TED	0.0733560266651933

Figure 14 is a graphical representation of the results of this function. The line widths of the nodes reflect pagerank.

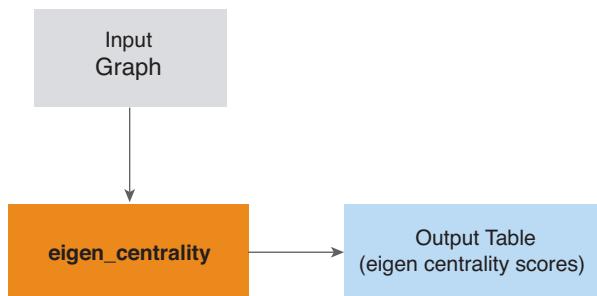
Figure 14: Pageranks in the Graph



eigen_centrality (beta)

Summary

This function computes eigenvector centrality for a directed graph.



Background

Eigenvector centrality is a measure of the influence of a node in a network. An algorithm implementing eigenvector centrality assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.

The sum of squared eigenvector centrality scores is 1.

Usage

Syntax (version 1.0)

```
SELECT * FROM EIGEN_CENTRALITY (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')

    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
    [ MAXITERNUM('max_iter') ]
    [ THRESHOLD('threshold') ]
    [ DELIMITER('delimiter') ]
) ;
```

Arguments

DOMAIN	Optional	The Aster Database queen's IP address or hostname. Has the form <i>host:port</i> . To specify an IPv6 address, enclose the host argument in square brackets (for example, <code>[::1]:2406</code>). The <i>port</i> is the port number that the queen is listening on. The default is the Aster Database standard port number (2406). For example, <code>DOMAIN(10.51.23.100:2406)</code> .
DATABASE	Optional	The name of the database where the input table resides. The default database is beehive.
USERID	Optional	The Aster Database user name of the user running this function. The default value is beehive.
PASSWORD	Required	The Aster Database user password.
INPUTTABLE	Required	The raw edges table. To avoid incorrect results when running this function, make sure there are no duplicate entries in the input table.
OUTPUTTABLE	Required	The name of the output table. If it already exists, an error is thrown.
STARTNODE	Optional	The name of the start node column. If not specified, the first column is picked.
ENDNODE	Optional	The name of the end node column. If not specified, the second column is picked.

<i>MAXITERNUM</i>	Optional	The maximum number of iterations the algorithm goes through. If not specified, 20 is used. This argument helps you prevent the function from running for a long period time when dealing with large graphs. For example, when dealing with a very large graph, you could choose to let the algorithm implemented by this function to iterate through the graph for 10 times. If the algorithm does not converge (the results are not stable), you can increase it until the algorithm converges.
<i>THRESHOLD</i>	Optional	The algorithm convergence criteria value. If not specified, 0.0001 is used. The algorithm converges when the results are stable and don't change too much.
<i>DELIMITER</i>	Optional	Allowed values are the space character (' '), '#', '\$', '%', '&'. The default is '' (space). This delimiter is used to concatenate nodes as a string. Any node should not have a delimiter as one of its characters.

Example

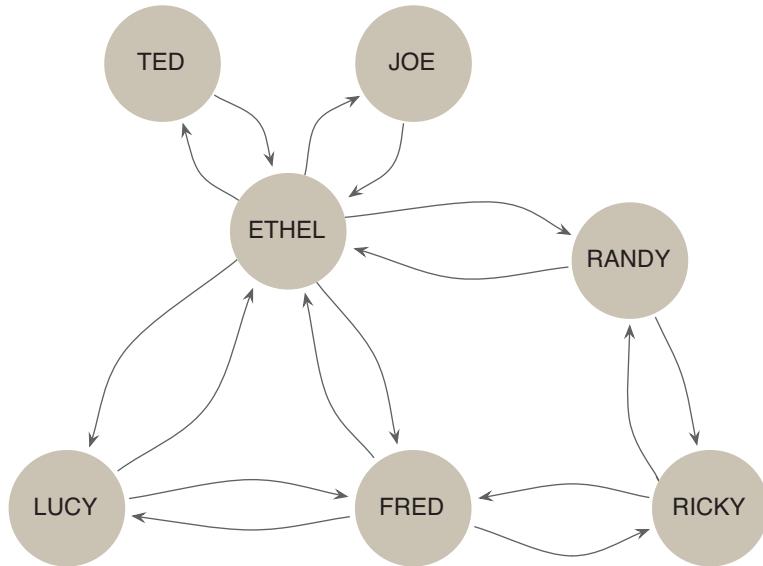
Sample Input

Table 11 - 299: Example input table: raw_edges

vertex1	vertex2
TED	ETHEL
ETHEL	TED
RICKY	FRED
FRED	RICKY
ETHEL	LUCY
LUCY	ETHEL
ETHEL	RANDY
RANDY	ETHEL
FRED	ETHEL
ETHEL	FRED
JOE	ETHEL
ETHEL	JOE
RANDY	RICKY
RICKY	RANDY
LUCY	FRED
FRED	LUCY

[Figure 15](#) is a graphical representation of the graph described by the raw_edges table.

Figure 15: Input Graph



Sample SQL-MR call:

```
SELECT * FROM EIGEN_CENTRALITY (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('beehive')
    USERID('beehive')
    PASSWORD('beehive')
    inputTable('raw_edges')
    outputTable('eigen_centrality_string')
    threshold('0.01')
) ;
```

Sample output

The eigen_centrality function displays these Console messages:

Table 11 - 300: eigen_centrality Console Messages

message
Total non-null nodes number: 7.
Non-null end nodes number: 7.
sq_err
7.0
0.254256878112061
0.117499130462478
0.0713652363850385
0.0480245355572354

Table 11 - 300: eigen_centrality Console Messages (continued)

message
0.0336766665417
0.0239861638085265
0.0171786755952231
0.0123253933466242
0.00884758425523244
Successful. Algorithm converged.
Total iteration number: 10
The result is stored in table 'eigen_centrality_string'.

The last message in the Console output specifies the name of the table where the eigen_centrality function stores the results. To display the results, run this SQL query:

```
SELECT * FROM eigen_centrality_string
ORDER BY eigen_centrality DESC;
```

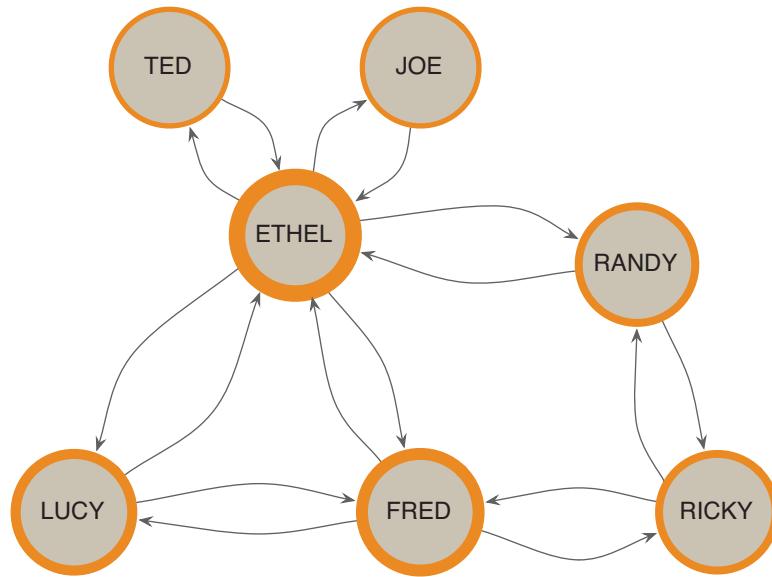
This query lists the graph nodes and their corresponding eigen_centrality scores.

Table 11 - 301: Eigenvector Centrality Scores

node	eigen_centrality
ETHEL	0.618244460188595
FRED	0.449438527641803
LUCY	0.382575098561845
RANDY	0.304858475196445
RICKY	0.304083247033199
TED	0.204950945658131
JOE	0.204950945658131

Figure 16 is a graphical representation of the eigenvector centrality scores in this example. The line widths of the nodes reflects the scores.

Figure 16: Eigenvector Centrality Scores



Tip: This function does not generate graphical representation of the results. This graphic was generated manually.



CHAPTER 12 Data Transformation

- [Antiselect](#)
- [Multicase](#)
- [Pack](#)
- [Unpack](#)
- [Pivot](#)
- [Unpivot](#)
- [XMLParser](#)
- [XMLRelation](#)
- [JSONParser](#)
- [Apache Log Parser](#)
- [outlierFilter](#)
- [IpGeo](#)

Antiselect

Summary

Antiselect returns all columns *except* those specified in the exclude clause.



Background

There are cases in which a user wants to retrieve all but a few columns from a table or query. The syntax rules of SELECT allow you to either specify "*" or list the specific columns you want, but you cannot list columns to be excluded.

With the *antiselect* function, you can select all columns except the one(s) in the EXCLUDE argument clause list. This is useful when simply selecting rows or when doing a join to create a new table.

Usage

This section describes the syntax for using the function, parameter options and data types, and a description of the expected output.

Syntax (version 1.0)

```
SELECT *
  FROM antiselect
  (
    ON {table_name | view_name | (query)}
    EXCLUDE('column_name' [, ...])
  );
```

Arguments

<i>EXCLUDE</i>	Optional Specifies the list of the columns that will not be returned. The list is specified as a comma-delimited list of column names, each in single-quotes.
----------------	---

Output

The function emits all columns except those named in the EXCLUDE list.

Example

Example Input Data

Table 12 - 302: Example input table: antiselect_test

id	src	age	gender	race	numBuys	numSells
1	ebay	62	male	white	30	44
2	paypal	29	female	asian	33	23

Example SQL-MapReduce Call

This example runs on the table *antiselect_test* and excludes the “race” column from its output:

```
SELECT *
  FROM antiselect
  (
    ON antiselect_test
    EXCLUDE('race')
  );
```

Example Output of Antiselect

Table 12 - 303: Example output table

id	src	age	gender	numBuys	numSells
1	ebay	62	male	30	44
2	paypal	29	female	33	23

Multicase

Summary

Multi-case extends the capability of the SQL CASE statement by supporting matches to multiple criteria in a single row. The function iterates through the input data set only once and emits matches whenever a match occurs. If multiple matches occur for a given input row, one output row will be emitted for each match. This differs from the behavior of the SQL CASE statement. When SQL CASE finds a match, it emits the result and immediately proceeds to the next row without searching for more matches in the current row.



Background

The multicase function is useful when you want to have a single row match multiple conditions. In other words, you should use this function when the conditions in your CASE statement do not form a mutually exclusive set.

Usage

Syntax (version 1.1)

Below is how the multi-case is invoked using a SELECT statement to invoke the SQL-MapReduce function.

```

SELECT *
  FROM multi_case
  (
  ON
  (
    SELECT * ,
      Condition1 AS case1,
      Condition2 AS case2,
      ....
      ConditionN AS caseN
    FROM {table_name|view_name|(query)}
  )
  LABELS
  (
    'case1 AS "label1"',
  )
  
```

```

    'case2 AS "label2"',
    ...,
    'caseN AS "labelN"'
)
);

```

Arguments

<i>CONDITIONS</i>	Each condition is an SQL predicate that evaluates to true or false.
<i>LABELS</i>	For each case, you must specify a label the function will apply to matches of that case. Specify this in the <i>LABELS</i> clause in the form, 'case1 AS "label1"'.

Input

Input rows should contain at least one column.

Output

A row is output for each match. All input columns are conveyed as-is to the output, and a category name column is added.

Example

Example Input Data

Table 12 - 304: Example Input Data, table people

userid	name	age
100	Henry Cavendish	12
200	Sir William	15
300	Johann August	19
400	Martin Heinrich	20
500	Ralph Arthur	25
600	Marguerite Catherine	35
700	Philip Hauge	40
800	Joseph Louis	28
900	Marie Curie	12

Example SQL-MapReduce call

```

SELECT *
FROM multi_case
(
ON
(
SELECT *,  

    (age < 1) AS case1,  

    (age >= 1 AND age <= 2) AS case2,

```

```

        (age >= 2 AND age <=12) AS case3,
        (age >=13 AND age <=19) AS case4,
        (age >=16 AND age <=25) AS case5,
        (age >=21 AND age <=40) AS case6,
        (age >=35 AND age <=60) AS case7,
        (age >=60) AS case8
    FROM people
)
LABELS
(
    'case1 AS "infant"',
    'case2 AS "toddler"',
    'case3 AS "kid"',
    'case4 AS "teenager"',
    'case5 AS "young adult"',
    'case6 AS "adult"',
    'case7 AS "middle aged person"',
    'case8 AS "senior citizen"'
)
)
ORDER BY userid;

```

Example Output from Multicase

Table 12 - 305: Example output table

UserID	Name	Age	Category
100	Henry Cavendish	12	kid
200	Sir William	15	teenager
300	Johann August	19	teenager
300	Johann August	19	young adult
400	Martin Heinrich	20	young adult
500	Ralph Arthur	25	young adult
500	Ralph Arthur	25	adult
600	Marguerite Catherine	35	adult
600	Marguerite Catherine	35	middle aged person
700	Philip Hauge	40	adult
700	Philip Hauge	40	middle aged person
800	Joseph Louis	28	adult
900	Marie Curie	12	kid

Pack

Summary

The *Pack* function takes a number of input columns and transforms them into a single *packed column*. The packed column is composed of a number of *virtual columns*, with each virtual column representing an input column. In the packed column, a *COLUMN_DELIMITER* string separates each virtual column value from the next. By default, each virtual column value is labeled with its column name, but you can turn off this labeling.



Aster Database also provides an *Unpack* function to explode a single packed column into a number of columns. See “[“Unpack” on page 471](#)”.

Usage

Syntax (version 1.1)

```

SELECT *
  FROM pack
  (
    ON { table_name|view_name|(query) }
    [COLUMN_NAMES('column1' [, ...])]
    [COLUMN_DELIMITER('delimiter_value')]
    [INCLUDE_COLUMN_NAME('true'|'false')]
    PACKED_COLUMN_NAME('packed_column_name')
  );
  
```

Arguments

ON	Required	Table, view, or query that provides the data to be packed.
COLUMN_NAMES	Optional	Names of the input columns to be packed. Format these as a comma-delimited list with each name in single-quotes. The name of each input column that you pack becomes its virtual column name. Columns that are present in the input but not listed in COLUMN_NAMES are passed through to the output as regular columns. If you do not include a COLUMN_NAMES clause, all input columns are packed, and no columns are passed through as-is.
COLUMN_DELIMITER	Optional	Character string that will be used to delimit each packed data value (and virtual column name if present) from the next. If the column delimiter is a pipe character (that is, ' '), you need to double escape it (that is, '\\ '). The default is a comma (','). Note that this can be more than a single character but a regular expression (as is allowed in the Aster Database <i>unpack</i> function).
INCLUDE_COLUMN_NAME	Optional	A true or false value that specifies whether or not to pre-pend each packed value with its virtual column name. The default is <i>true</i> .
PACKED_COLUMN_NAME	Required	Name of the output column that will hold the packed data.

Output

The packed data column (this holds the virtual columns), as well as any input columns (in as-is, unpacked condition) that you did not name in *COLUMN_NAMES*. The packed data column is of type *varchar*.



Important! Each time you pack a table, make a note of the datatypes of all its packed columns. You will need to know these types later, if you wish to unpack them.

Example:

Example Input Data

Table 12 - 306: Example Input Data, table pack_test

id	src	age	gender	race	numbuys	numsells
1	ebay	62	male	white	30	44
2	paypal	29	female	asian	33	23

Example SQL-MapReduce call

```
SELECT *
  FROM  pack
  (
    ON  pack_test
    COLUMN_DELIMITER(' , ')
    PACKED_COLUMN_NAME('packed_data')
    INCLUDE_COLUMN_NAME('true')
    COLUMN_NAMES('src', 'age', 'gender', 'race', 'numbuys', 'numsells')
  );
```

In the example query above, note that we did not include the “id” column in the *COLUMN_NAMES* list, even though it was one of the input columns. This has the effect of passing the id column to the output in its original, unpacked condition.

Example Output from Pack

Table 12 - 307: Example Output from Pack

packed_data	id
numbuys:30,age:62,gender:male,numsells:44,src:ebay,race:white	1
numbuys:33,age:29,gender:female,numsells:23,src:paypal,race:asian	2

Unpack

Summary

The Unpack function takes data from a single *packed column* and expands it to multiple columns. Each packed column is composed of a number of *virtual columns*. In the packed column, a *COLUMN_DELIMITER* string separates each virtual column from the next.



This function is complementary to Aster Database's [Pack](#) function, but can be used on any packed column that represents its packed columns in a reasonably regular way. See “[Pack](#)” on page 469.

Usage

Syntax (version 1.1)

```

SELECT *
  FROM unpack
  (
    ON {table_name | view_name | (query)}
    DATA_COLUMN('data_column')
    COLUMN_NAMES('column1' [, 'column2', ...])
    COLUMN_TYPES('datatype' [, 'datatype', ...])
    [COLUMN_DELIMITER('delimiter_value')]
    [DATA_PATTERN('data_pattern_regular_expression')]
    [DATA_GROUP('group_number')]
    [IGNORE_BAD_ROWS({'true' | 'false'})]
  );
  
```

Arguments

<i>DATA_COLUMN</i>	Required	Name of the input column that contains the packed data to be unpacked.
<i>COLUMN_NAMES</i>	Required	Names to be given to the output columns, specified as a comma-delimited list. These are the columns that will be unpacked. You must list them in the order in which the virtual columns appear in your <i>DATA_COLUMN</i> .
<i>COLUMN_TYPES</i>	Required	The datatypes of the unpacked output columns, in the same order as the <i>COLUMN_NAMES</i> .
<i>COLUMN_DELIMITER</i>	Optional	The string that separates each virtual column from the next in the packed data. Note that if you choose a pipe delimiter (that is, ' '), you need to double escape it (that is, '\\ '). The default column delimiter is a comma (",").

<i>DATA_PATTERN</i>	Optional	<p>This is a regular expression that tells the Unpack function which part the packed data is the <i>actual data value</i>. When unpacking a virtual row of data, the virtual row consists of one unit of data for each virtual column, with each unit delimited from the next by a <i>COLUMN_DELIMITER</i>. Within each unit, in addition to the actual data value, there is often other information such as the virtual column name. The <i>DATA_PATTERN</i> allows Unpack to find the data value.</p> <p>In the <i>DATA_PATTERN</i>, you write regular expressions to identify the various regular parts of a virtual unit of data, and you use parentheses to surround the regular expression that matches the <i>actual data value</i>. The rest of each unit of data will be ignored.</p> <p>For example, let's assume we have packed data with two virtual columns, one for the <i>age</i> and one for the <i>gender</i> of a person. One example row of our packed data might look like this:</p> <pre>age : 34 , sex: male</pre> <p>To unpack this example, we would need to specify a <i>COLUMN_DELIMITER</i> of ',' and a <i>DATA_PATTERN</i> of ". * : (. *)". In the <i>DATA_PATTERN</i>, the first three characters, ". * :", are a standard regular expression (the "." is the wildcard that matches any character, and the "*" indicates the wildcard can be matched zero or more times) that matches both "age :" and "sex:", while the rest of the expression, "(. *)" is another regular expression that matches "34" and "male", with the parentheses instructing the unpack function to interpret these as the actual data values in the virtual column.</p> <p>The Unpack function's default <i>DATA_PATTERN</i> is "(. *)", which causes Unpack to recognize <i>as data</i> the entirety of each unit of data in the virtual row (for example the whole string between one <i>COLUMN_DELIMITER</i> and the next). Sticking with our first example, if we were to unpack using for the default data pattern of "(. *)", we'd get poor results because the Unpack function would return "age : 34" as a data value and "sex: male" as the next data value.</p> <p>Optionally, you can use multiple pairs of parentheses in your <i>DATA_PATTERN</i> to mark multiple <i>data groups</i> within the <i>DATA_PATTERN</i>, and then specify a <i>DATA_GROUP</i> number to indicate which <i>data group</i> is the <i>actual data value</i>.</p>
<i>DATA_GROUP</i>	Optional	<p>An integer counter value that specifies which <i>data group</i> in your <i>DATA_PATTERN</i> represents the <i>actual data value</i> in the virtual column. Recall from the preceding section that you use parentheses in your <i>DATA_PATTERN</i> to mark data groups in the pattern, and, by default, the Unpack function takes the last data group in each pattern to be the actual data value (other data groups are assumed to be virtual column names or unwanted data). If you want to use a data group other than the last one as your <i>actual data value</i>, then you must specify a <i>DATA_GROUP</i> value.</p> <p>For example, let's assume our <i>DATA_PATTERN</i> is:</p> <pre>([a-zA-Z]*) : (. *)</pre> <p>In this case, if we set <i>DATA_GROUP</i> to "1", then the string that matches ([a-zA-Z]*) will be unpacked as the actual data value. If we set <i>DATA_GROUP</i> to "2", then the string that matches (. *) will be unpacked as the actual data value.</p>
<i>IGNORE_BAD_ROWS</i>	Optional	<p>A true or false value that specifies whether the function will fail upon encountering a row with bad data (if 'false'), or ignore the bad row and proceed to the next row (if 'true'). Default is 'false'.</p>

Output

Output columns include the virtual columns you listed in *COLUMN_NAMES*, plus the other columns of the input table.

Example

Example Input Data

Table 12 - 308: Example input table: unpack_test

id	src	packed_data
1	ebay	62,male,white,30,44
2	paypal	29,female,asian,33,23
3	Bad_data	THISISINVALIDDATA

Example SQL-MapReduce call

```
SELECT *
  FROM unpack
  (
    ON unpack_test
    DATA_COLUMN('packed_data')
    COLUMN_NAMES('age', 'gender', 'race', 'numBuys', 'numSells')
    COLUMN_TYPES('integer', 'varchar', 'varchar', 'integer', 'integer')
    COLUMN_DELIMITER(',')
    DATA_PATTERN('(.*)')
    DATA_GROUP(1)
    IGNORE_BAD_ROWS('true')
  )
ORDER BY id;
```

Example Output from Unpack

Table 12 - 309: Example Output from Unpack

age	gender	race	numBuys	numSells	id	src
62	male	white	30	44	1	ebay
29	female	asian	33	23	2	paypal

Pivot

Summary

The pivot function is used to pivot data stored in rows into columns.

The function takes as input a table of data to be pivoted, and it automatically constructs the output schema based on the arguments passed to the function. NULL values are handled automatically by the function, as shown in the examples below.



Usage

Permissions

You must grant EXECUTE on the function “pivot” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```

SELECT * FROM pivot(
    ON {table_name | view_name | (query)}
    PARTITION BY col1[, col2, ...]
    [ ORDER BY order_by_columns ]
    PARTITIONS('col1'[, 'col2', ...])
    ROWS('number_of_rows')
    PIVOT_KEYS('key1', 'key2'[, ...])
    PIVOT_COLUMN( 'pivot_column_name' )
    METRICS('metric_col1', 'metric_col2'[, ...])
) ;
  
```

Arguments

<i>PARTITIONS</i>	Required	This should be the same as the columns after the 'PARTITION BY' clause, but can be in different order.
<i>ROWS</i>	Required	Either the 'ROWS' argument or the 'PIVOT_KEY' and the 'PIVOT_COLUMN' arguments need to be specified. The 'ROWS' argument specifies the maximum number of rows in all of the partitions. If the number of rows in a partition is smaller than this argument, then NULLs will be added; while if it is larger, the rest of the rows are omitted. See example 1 below for detail.

<i>PIVOT_KEYS & PIVOT_COLUMN</i>	Required	If the 'rows' argument is not specified, these two arguments must be specified together. All rows containing a value in the 'pivot_column' that is not specified as a pivot key will be ignored. If the partition does not contain a value for a particular pivot key, the function will emit NULL. Note that if you specify these two arguments, you must order the input rows lexicographically along the 'pivot_column'. If the pivot column contains numeric types, then a cast to varchar is required for the function to work properly. See example 2 below for detail.
<i>METRICS</i>	Required	The columns that contain the values you want to pivot.

Example 1

Suppose a table is made up of the columns member_id, week, value1 and value2. A user wants to generate a new table that has columns member_id, value1_0, value2_0, value1_1, value2_1, value1_2, and value2_2, based on scanning a specific set of 3 weeks from the original table.

Getting the data transformed in this way might be convenient for various analyses. The user can either specify the maximum number of rows to be pivoted in a certain partition, or specify particular rows to be pivoted by providing some 'pivot keys'.

Constructing this table using standard SQL would require multiple self-joins using outer joins, and would be inefficient. Example 1 shows usage of the function when the 'rows' argument clause is specified.

Example 1 Input Data

Table 12 - 310: Example input table: pivot_test_1

member_id	week	value1	value2
1	1	100	1000
1	2	103	1030
1	3	107	1070
2	1	202	2020
2	3	205	2050

Example 1 SQL-MapReduce Call

```
SELECT * FROM pivot(
    ON pivot_test_1
    PARTITION BY member_id
    ORDER BY week
    PARTITIONS ('member_id')
    ROWS (3)
    METRICS ('value1', 'value2')
) ORDER BY member_id;
```

Example 1 Output from Pivot

Table 12 - 311: Example output table

member_id	value1_0	value2_0	value1_1	value2_1	value1_2	value2_2
1	100	1000	103	1030	107	1070
2	202	2020	205	2050		

Notice that the number of rows in partition 'member_id = 2' is 2, so the third set of values for this partition are NULLs, as is shown in the output table. Also, for the notation value1_i, the 'value1' part corresponds to the column name in the input table, while the 'i' part corresponds to the order of rows in a partition. The 'ORDER BY' clause is not required, and if not supplied, the order of values is not assured. NULLs are added at the end.

Example 2

In this example, we specify the 'pivot_keys' to include as columns. We also need to specify the 'pivot_column' where the pivot keys exist, and must order the input rows lexicographically along the 'pivot_column'. The input data will be the "pivot_test_2", just as in Example 1.

Example 2 Input Data

Same as Example 1.

Example 2 SQL-MapReduce call

```
SELECT * FROM pivot(
    ON pivot_test_1
    PARTITION BY member_id
    ORDER BY week::varchar
    PARTITIONS ('member_id')
    PIVOT_KEYS('2', '3')
    PIVOT_COLUMN('week')
    METRICS ('value1', 'value2')
)
ORDER BY member_id;
```

Example 2 Output from pivot

Table 12 - 312: Example 2 Output from pivot

member_id	value1_2	value2_2	value1_3	value2_3
1	103	1030	107	1070
2			205	2050

In this case, the PIVOT_KEYS are '2' and '3', so the rows where 'week = 1' are not included. And for pivot key '2', member 2 does not have any data for the row where 'week = 2', so NULLs are added in the output table. Note how the notation is constructed: for example, for the notation value1_i, the 'value1' part still refers to the column name in the input table, while the 'i' part refers to the pivot_keys.

Unpivot

Summary

Unpivot is a SQL-MR function to convert columns into rows.



Background

This is a “utilities” function that is the inverse of the pivot SQL-MR function.

Usage

Syntax (version 1.1)

```

SELECT * FROM unpivot
(
    ON input_timeseries_table
    [COLSTOUNPIVOT('column1', 'column2', ...)] |
    [COLSTOUNPIVOTRANGE('['index1:index2]', '['index3:index4]', ...)]
    COLSTOACCUMULATE('column1', 'column2', ...)
    [KEEPINPUTCOLUMNTYPES('true|false')]
    [ATTRIBUTECOLUMNNAME('column_name1')]
    [VALUECOLUMNNAME('column_name2')]
) ;
  
```

Arguments

<i>input_timeseries_table</i>	Required	The input table to be unpivot.
<i>COLSTOUNPIVOT</i>	Optional	The name of the columns to convert to rows. NOTE: If you do not specify this argument, then you must specify the <i>COLSTOUNPIVOTRANGE</i> argument.
<i>COLSTOUNPIVOTRANGE</i>	Optional	The ranges of columns to convert to rows. NOTE: If you do not specify this argument, then you must specify the <i>COLSTOUNPIVOT</i> argument.
<i>COLSTOACCUMULATE</i>	Required	The names of the columns in the input table to keep in the output table. The order in which you specify the column names in this clause should match the order of the corresponding columns in the input table. For example, assume the input table has these columns: ID1, ID2, SENS1, SENS2, SENS3. If you want to accumulate columns ID1 and ID2, use <i>colsToAccumulate('id1','id2')</i> and <i>not colsToAccumulate('id2','id1')</i> .
<i>KEEPINPUTCOLUMNTYPES</i>	Optional	Whether to keep the types of the input columns. The two possible values are true or false.
<i>ATTRIBUTECOLUMNNAME</i>	Optional	The unpivoted attribute column name. The default name is ATTRIBUTE
<i>VALUECOLUMNNAME</i>	Optional	The unpivoted value column name. The default name is VALUE.

Input

The unpivot function takes as input a table whose columns you want to convert into rows.

Output

The unpivot function generates an output table with the specified columns in the input table converted into rows, as shown in the examples below.

Examples

Example 1

This example shows usage of the function in its current form.

Input

Table 12 - 313: Input table (inputDataToUnpivot)

ID1	ID2	SENS1	SENS2	SENS3
5	2	19	9581	15.6246871106
8	4	43	6830	10.8187293343
1	1	74	5697	17.3705606244
5	3	78	8139	14.3446582505

Sample SQL-MR Call

```
SELECT * FROM Unpivot(
    ON inputDataToUnpivot
    colsToUnpivot('sens1','sens2','sens3')
    colsToAccumulate('id1','id2')
)
ORDER BY id2;
```

Output

Table 12 - 314: Output table

ID1	ID2	ATTRIBUTE	VALUE
1	1	SENS1	74
1	1	SENS2	5697
1	1	SENS3	17.37056
5	2	SENS1	19
5	2	SENS2	9581
5	2	SENS3	15.624687
5	3	SENS1	78
5	3	SENS2	8139

Table 12 - 314: Output table

ID1	ID2	ATTRIBUTE	VALUE
5	3	SENS3	14.344658
8	4	SENS1	43
8	4	SENS2	6830
8	4	SENS3	10.818729

Example 2

This example uses the same input table used in Example 1 ([Table 12 - 313](#)) and uses the `keepInputColumnTypes('true')` option, which supports 3 different input column types:

- Real numbers (float/double) outputted as double
- Integers (int, short, long) outputted as long
- Anything else outputted as varchar (string)

Sample SQL-MR Call

```
SELECT * FROM Unpivot(
    ON inputDataToUnpivot
    colsToUnpivot('sens1', 'sens2', 'sens3')
    colsToAccumulate('id1', 'id2')
    keepInputColumnTypes('true')
)
ORDER BY id2;
```

Output

Table 12 - 315: Output table

ID1	ID2	ATTRIBUTE	VALUE_LONG	VALUE_DOUBLE	VALUE_STR
1	1	SENS1	74		
1	1	SENS2	5697		
1	1	SENS3		17.3705596923828	
5	2	SENS1	19		
5	2	SENS2	9581		
5	2	SENS3		15.6246871948242	
5	3	SENS1	78		
5	3	SENS2	8139		
5	3	SENS3		14.3446578979492	
8	4	SENS1	43		
8	4	SENS2	6830		
8	4	SENS3		10.8187294006348	

Example 3

You can also specify a range of column indices that you would like to unpivot instead of the column names. In this example, the unpivot function has the identical output as in the previous example.

Sample SQL-MR Call

```
SELECT * FROM Unpivot (
    ON inputDataToUnpivot
    colsToUnpivotRange('[2:4]')
    colsToAccumulate('id1','id2')
    keepInputColumnTypes('true')
) ORDER BY id2;
```

Note that the first column index starts at 0, therefore SENS1, SENS2, and SENS3 are the index number 2, 3, and 4 respectively.

Output

Same as [Table 12 - 313](#).

Example 4

You can also specify a list of range indices as in this example.

Sample SQL-MR Call

```
SELECT * FROM Unpivot (
    ON inputDataToUnpivot
    colsToUnpivotRange('[2:3],[4:4]')
    colsToAccumulate('id1','id2')
    keepInputColumnTypes('true')
) ORDER BY id2;
```

Output

Same as [Table 12 - 313](#).

Error Messages

Unpivot returns these error messages:

- ERROR: SQL-MR function UNPIVOT failed: INTERVALS should be in the form of [<min>:<max>]. <min> should be less or equal to <max>.
REASON: The index pairs in colsToAccumulateRange are not in ascending order (or equal). For example, index1 > index2.
- ERROR: SQL-MR function UNPIVOT failed: <index2> in "colsToUnpivotRange('[index1:index2]'" cannot exceed or be equal to the index column size of the input table (5) where (5) in this case is the size of the input table (column range)
REASON: One of the indices exceeds the column range.
- ERROR: SQL-MR function UNPIVOT failed: "colsToAccumulate" must be different from "colsToUnpivot"
REASON: colsToAccumulate (or colsToAccumulateRange) has common columns with the colsToUnpivot argument clause.

XMLParser

Summary

The *XMLParser* function is a general-purpose tool for extracting information from XML documents. This function extracts element names, attribute values, and text.



Background

XML data is semi-structured and does not conform with the formal structure of data in relational database tables. XML data is hierarchical, which is not the case in relational database tables. As a result, XML data cannot be searched and analyzed using SQL queries.

To address this issue, you can use the *XMLParser* function which lets you extract the relevant elements from input XML data and *relationalize* them. The output of this function is a *relational* table, which you can query using SQL. The *XMLParser* function lets you specify the schema of the output table.

Because not all kinds of XML data can be converted into a table, the *XMLParser* function constrains the relationships in the extracted data to be parent/child or sibling relationships. The *XMLParser* function lets you specify these constraints.

Usage

This function reads the entire input XML data into memory. Make sure that the size of the input data does not exceed the available memory on your system.

Permissions

You must grant EXECUTE privileges to the database user who runs this function. For more information, see “[Set Permissions to Allow Users to Run Functions](#)” on page 53.

Syntax (version 1.3)

```

SELECT * FROM XMLParser(
    ON { table_name | view_name | (query) }
    TEXT_COLUMN('text_column_name')
    NODES('node_pair_string[,...]')
    [SIBLING('sibling_node_string[,...]')
    [DELIMITER('delimiter_string')]
    [SIBLING_DELIMITER('delimiter_string')]
    [MAX_ITEM_NUMBER('max_item_number')]
    [ANCESTOR('nodes_path')]
    [OUTPUTCOLUMN_NODEID('column_name')]
    [OUTPUTCOLUMN_PARENT_NODE_NAME('column_name')]
    [ERROR_HANDLER('false|true[; [output_column_name:] column_names]')]
    [ACCUMULATE('column [, ...]')])
);

```

Arguments

<i>TEXT_COLUMN</i>	Required	Name of the column containing the input XML data. Only one column is permitted.
<i>NODES</i>	Required	A list of the parent/child node-pair strings from which the function extracts data. For each node, you can specify the attributes to extract. The <i>XMLEParser</i> function generates a row for each node-pair string in the output table and adds a column for each attribute you specify in this argument. <ul style="list-style-type: none"> If the parent node has only one child, use this syntax: <code>NODES ('parent_node_name[:attributes] / child_node_name[:attributes]')</code> If the parent node has multiple child nodes, use this syntax: <code>NODES ('parent_node_name[:attributes] / {child_node_name[:attributes], child_node_name[:attributes], ...}')</code> To specify one attribute, use this syntax in place of <i>attributes</i>: <code>node_name:attribute_name</code> To specify multiple attribute, use this syntax in place of <i>attributes</i>: <code>node_name:{attribute_name, ...}</code> In addition to exact matching, parent nodes without attributes can contain wildcards for fuzzy matching. <i>XMLEParser</i> supports two kinds of wildcard matching: <ul style="list-style-type: none"> SQL Like: You can use rules similar to SQL Like when matching the parent node. The string, which is input by user and stands for a parent node name, should follow this format: '<code>like (expression)</code>'. Three wildcards are supported here: <ul style="list-style-type: none"> A percent sign (%) matches any sequence of zero or more characters. An underscore (_) matches any single character. An escape character (\) is used when a percent, underscore, or the "escape character" itself is required in the pattern without its special meaning. For example, <code>NODES ('like (%a_c_) /d')</code> matches the XML fragment "<code><123abc_><d>text</d></123abc_></code>". Java Regular Expression: Java regular expression rules can also be used when matching the parent node. In this case, the string of parent node name input by the user should follow the format '<code>regex (expression)</code>'. NOTE: In the <i>NODES</i> argument, only the parent node without attributes can be wildcard-matched. Children nodes and parent nodes with attributes can only be exact-matched.
<i>SIBLING</i>	Optional	A list of the sibling nodes of one of the parent nodes specified in the <i>NODES</i> argument. The <i>XMLEParser</i> function includes the values from the sibling nodes in every output row and adds a column in the output table for every sibling node and every attribute specified by this argument. In addition, <i>XMLEParser</i> adds a column for every attribute specified in this argument. To specify a list of sibling nodes, use this syntax: <code>SIBLING ('node_name[:attributes] [, ...]')</code>
<i>DELIMITER</i>	Optional	The delimiter that separates multiple child node values in the output. If not defined, <i>XMLEParser</i> uses the comma (",").
<i>SIBLING_DELIMITER</i>	Optional	The delimiter that separates multiple sibling node values in the output. If not defined, <i>XMLEParser</i> uses the comma (",").

<i>MAX_ITEM_NUMBER</i>	Optional	The maximum number of sibling nodes with the same name to be returned. This should be a positive integer less than or equal to 10 (<=10). The default value is 10.
<i>ANCESTOR</i>	Optional	The path of the ancestor nodes of all the parent nodes specified in the <i>NODES</i> argument. If no ancestor path is defined, the root of the XML document is used as the default value. Use this syntax to specify the ancestor path: <code>ANCESTOR ('node_name[:attributes]')</code> In addition to exact matching, ancestor nodes without attributes can contain wildcards for fuzzy matching. Two kinds of wildcard matching are supported here, which are exactly the same as the two kinds of matching described in the <i>NODES</i> argument description above.
<i>OUTPUTCOLUMN_NODEID</i>	Optional	Name of the column in the output table that contains the ID of each node the <i>XMLParser</i> function has extracted. The default name is <i>out_nodeid</i> .
<i>OUTPUTCOLUMN_PARENT_NODE_NAME</i>	Optional	Name of the column in the output table that the <i>XMLParser</i> function uses to store the tag names of the extracted parent nodes. The default name is <i>out_parent_node</i> .
<i>ERROR_HANDLER</i>	Optional	Error handler for handling the errors that occur when parsing an XML document. The default value is 'false'. In the default case (<i>ERROR_HANDLER('false')</i>), if a parsing error occurs, the function aborts and throws an exception. If you enable error handling (<i>ERROR_HANDLER('true')</i>), if an error occurs while parsing a row, the function skips that row and goes to the next row. When the function completes the parsing, it outputs the nodes in the XML document that were error free. When you enable error handling, you can also specify that additional information be outputted about the errors. For example, if you use this clause: <code>ERROR_HANDLER('true;error_info:column1, column2')</code> the function adds the column "error_info" to the output table and adds to it the content of the input columns "column1" and "column2," separated by a semicolon. If you enable error handling without specifying an output column name, for example, <i>ERROR_HANDLER('true;column1, column2...')</i> , the default column name is "error_handler."
<i>ACCUMULATE</i>	Optional	A list of the column names you want to add to the output table. The names you specify here cannot be the same as the names of the columns specified in the <i>OUTPUTCOLUMN_NODEID</i> or <i>OUTPUTCOLUMN_PARENT_NODE_NAME</i> arguments. By default, if you do not use this argument, the <i>XMLParser</i> function returns all input columns.

Input Schema

The *XMLParser* function does not check the schema of the input XML data.

Malformed Documents

If malformed XML documents are encountered, the function skips them and continue to parse the next document.

Output Schema

The *XMLEParser* function generates a row for each node specified in the *NODES* argument and for each of descendants of the ancestor path indicated in the *ANCESTOR* argument.

The output table contains the following columns:

- One column for the node ID
- One column for the name of the parent node
- One or more columns for each specified parent attribute
- One or more columns for each specified sibling
- One or more columns for each specified sibling attribute
- One or more columns for each specified child node
- One or more columns for each specified child node attribute
- One or more columns for each specified ancestor attribute
- All columns specified in the *ACCUMULATE* argument

No Output Conditions

The *XMLEParser* function does not generate output in these cases:

- There is no ancestor path in the XML data as specified in the *ANCESTOR* argument.
- There are no parent nodes in the XML data as specified in the *NODES* argument.

Null Values in Rows

The *XMLEParser* function sets the values of row cells to null in these cases:

- There are no children nodes in the XML data as specified in the *NODES* argument.
- There are no sibling nodes in the XML data as specified in *SIBLING* argument.
- There are no attributes in the XML data as specified in the *ANCESTOR*, *NODES*, or *SIBLING* arguments.

Examples

Input XML Documents

The `xml_inputs` table contains the XML documents that are used as input to the `XMLParser` functions in examples 1 and 2.

Table 12-316: Example input table: `xml_inputs`

xid	xmldocument
1	<pre><bookstore> <owner>billy</owner> <book category="WEB"> <title lang="en">XQuery Kick Start</title> <author>James McGovern</author> <author>Per Bothner</author> <year edition="1">2003</year> <year edition="2">2005</year> <price> <member>49.99</member> <public>60.00</public> </price> <reference> <title>A</title> </reference> <position value="1" locate="east"></position> </book> <book category="CHILDREN"> <author>Wenny Wang</author> <price> <member>99.99</member> <public>108.00</public> </price> </book> </bookstore></pre>

Table 12-316: Example input table: xml_inputs (continued)

xid	xmldocument
2	<pre> <setTopRpt xsi:noNamespaceSchemaLocation="Set%20Top%2020Report%20.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <settopid type="string" length="5">ST789</settopid> <accountid type="string">8728</accountid> <zipcode type="string">94025</zipcode> <reportstamp type="dateTime">2009-10-03T12:52:06</reportstamp> <temperature> <read type="bigDecimal">46</read> </temperature> <storage> <used type="bigDecimal">98</used> <used type="bigDecimal">199</used> <used type="bigDecimal">247</used> <total type="bigDecimal">300</total> </storage> <feed> <feedstamp type="dateTime">2009-10-03T12:52:06</feedstamp> </feed> </setTopRpt></pre>

The `xml_inputs_fuzzy` table contains the XML documents that are used as input to the `XMLParser` function in example 3.

Table 12-317: Example input table: xml_inputs_fuzzy

xid	xmldocument
1	<pre> <bookstore> <owner> "Wenny" </owner> <items> <bookitem category="WEB"> <title lang="en">XQuery Kick Start</title> <author>James McGovern</author> <author>Per Bothner</author> <year edition="1">2003</year> <price> <member>49.99</member> <public>60.00</public> </price> <position value="1" locate="east"/> </bookitem> </items> </bookstore></pre>

Table 12-317: Example input table: xml_inputs_fuzzy (continued)

xid	xmldocument
2	<cdstore> <owner> "Ann" </owner> <items> <recorditem category="pop"> <title lang="en">Breathe</title> <singer>Yu Quan</singer> <year>2003</year> <price> <member>29</member> <public>35</public> </price> <position value="1" locate="east"/> </recorditem> </items> </cdstore>

The `xml_inputs_error` table contains the XML documents that are used as input to the `XMLParser` function in example 4.

Table 12-318: Example input table: xml_inputs_error

xid	xmldocument
1	<bookstore owner="Wenny"> <items> <bookitem category="WEB"> <title lang="en">XQuery Kick Start</title> <author>James McGovern</author> <author>Per Bothner</author> <year edition="1">2003</year> <price> <member>49.99</member> <public>60.00</public> </price> <position value="1" locate="east"/> </bookitem> </items> </bookstore>
2	<bookstore>

Example 1

SQL-MapReduce Call

```
SELECT * FROM XMLParser(
  ON xml_inputs
  TEXT_COLUMN('xmldocument')
  NODES ('price/member')
  SIBLING ('author')
  SIBLING_DELIMITER(';')
  ACCUMULATE('xid')
);
```

XMLParser Output

Table 12-319: Example output table

xid	out_node_id	out_parent_node	author	member
1	1	price	James McGovern; Per Bothner	49.99
1	2	price	Wenny Wang	99.99

Example 2

SQL-MapReduce Call

```
SELECT * FROM XMLParser(
    ON xml_inputs
    TEXT_COLUMN('xmldocument')
    NODES ('temperature/read:type','storage/{used, total}')
    SIBLING ('settopid:{type, length}', 'accountid')
    ANCESTOR('setTopRpt')
    OUTPUTCOLUMN_NODEID('nid')
    MAX_ITEM_NUMBER(1)) ;
```

XMLParser Output

Table 12-320: Example output table

nid	out_parent_node	settopid	settopid:type	settopid:length	accountid	read	read:type	used	total
1	temperature	ST789	string	5	8728	46	bigDecimal		
2	storage	ST789	string	5	8728			98	300

Example 3

SQL-MapReduce Call

```
SELECT * FROM XMLParser(
    ON xml_inputs_fuzzy
    TEXT_COLUMN('xmlDocument')
    NODES('like(%store)/owner','regex([a-z]+item)/{title,year}')
    ANCESTOR('like(%store)')
    ACCUMULATE('xid')
) ;
```

XMLParser Output

Table 12 - 321: Example output table

xid	out_node_id	out_parent_node	owner	title	year
1	1	bookstore	Wenny		
1	2	bookitem		XQuery Kick Start	2003
2	1	cdstore	Ann		
2	2	cditem		Breathe	2003

Example 4

SQL-MapReduce Call

```
SELECT * FROM XMLParser(
  ON xml_inputs_error
  TEXT_COLUMN('xmlDocument')
  NODES('bookstore:owner', 'bookitem/title')
  ERROR_HANDLER('true;xmlDocument')
  ACCUMULATE('xid')
) ;
```

XMLParser Output

Table 12 - 322: Example output table

xid	out_node_id	out_parent_node	bookstore:owner	title	error_handler
1	1	bookstore		Wenny	
1	2	bookitem		XQuery Kick Start	
2					<bookstore>;

Error Messages

XMLParser returns these error messages:

- ERROR: SQL-MR function XMLPARSER requires column: ***.
REASON: If needed columns are missing from the relation named in the ON clause.
- ERROR: SQL-MR function XMLPARSER failed: The column '***' already exists in the output column list.
REASON: If any columns in your ACCUMULATE clause have a prohibited column name.
- ERROR: SQL-MR function XMLPARSER failed: The format of '***' is incorrect.
REASON: If the format of the string NODES, SIBLING, or ANCESTOR are incorrect.
- ERROR: SQL-MR function XMLPARSER failed: SIBLING_DELIMITER argument can be used only with SIBLING argument.
REASON: If specified SIBLING_DELIMITER argument without SIBLING argument.
- ERROR: The maxItemNumber should be a positive integer.
REASON: If the value of MAX_ITEM_NUMBER is not a positive integer.
- ERROR: SQL-MR function XMLPARSER failed: The column '***' has already exist in the output column list.
REASON: If two output columns name are same, such as two children nodes with the same name in the output.
- ERROR: SQL-MR function XMLPARSER failed: The format of wildcard matching is regex|like(expression). Only nodes without attributes can be wildcard matched. Found: ***
REASON: If parent nodes in NODES or ancestor nodes in ANCESTOR contains malformed wildcard expressions, or they contain both attributes and wildcard matching expressions

XMLRelation

Summary

The XMLRelation function is a tool for extracting most XML content (element name, text and attribute values) and structural information from XML documents into a relational table.



Background

The XMLRelation function is a general tool for flattening XML documents into a relational table for multiple uses. It extracts most XML content and maintains multi-level paths for the XML elements from the input XML documents. It then flattens the data into a relational table. After performing this transformation, you can query the data by accessing the output table through SQL.

The inputs to XMLRelation function are the XML data, the parameter that specifies the maximum depth in the parsing process, and the parameters that specify the output data. The output of the XMLRelation function is a flattened table containing the data from the XML documents.

Usage

Permissions

You must grant EXECUTE on the function “XMLRelation” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.1)

```

SELECT *
FROM XMLRelation(
    TEXTCOLUMN ('text_column_name')
    DOCIDCOLUMNS ('docid_columns')
    [MAXDEPTH ('max_depth')]
    [EXCLUDEELEMENTS ('node_paths')]
    [ATTRIBUTETONODE ('false' | 'true')]
    [ATTRIBUTEDELIMITER ('delimiter_string')]
    [OUTPUT ('fulldata' | 'parentchild' | 'fullpath')]
    [ERROR_HANDLER('false' | 'true' [; [output_column_name:] column_names] )]
    [ACCUMULATE ('column_names')]
);

```

Arguments

<i>TEXTCOLUMN</i>	Required	Name of the column that contains the XML documents to be scanned. Only one column is permitted.
<i>DOCIDCOLUMNS</i>	Required	A list of the columns which make up the identifiers of the input XML documents. Note that no column name specified can be the same as a column name indicated in the Output schema.
<i>MAXDEPTH</i>	Optional	The maximum depth in the XML tree of the XML documents that will be processed. Note that the <i>MAXDEPTH</i> and <i>OUTPUT</i> arguments determine the schema of the output table, and the number of columns in the output table should not exceed 1600. Default is 5.
<i>EXCLUDEELEMENTS</i>	Optional	Used to supply a list of nodes to exclude as path strings. For each path string supplied, that node and all of its child nodes will be omitted from processing by the function. Path strings must follow the format ' <i><node1[.../nodeN]></i> ' or ' <i><node1/.../{nodeN[...]}></i> '. Brackets like {} are used to specify only particular nodes at that level, and are only allowed at the end of the path. For example, 'chapter' and 'root/book/{author,chapter}' are both legal paths.
<i>ATTRIBUTETONODE</i>	Optional	[true false] If set to 'false', attributes of one XML node are stored in one element of the output tuple. If set to 'true', the function will handle attributes as children of the node. Default is 'false'.
<i>ATTRIBUTEDELIMITER</i>	Optional	Specifies the delimiter used to separate multiple attributes of one node in XML documents. The default value is a comma ','.
<i>OUTPUT</i>	Optional	Specifies the schema of the output table. There are three options, 'fulldata', 'parentchild' and 'fullpath'. More details are shown in the XMLRelation Examples . Note that the <i>MAXDEPTH</i> and <i>OUTPUT</i> arguments determine the schema of the output table, and the number of columns in the output table should not exceed 1600. Default is 'fullpath'.
<i>ERROR_HANDLER</i>	Optional	Error handler for handling the errors that occur when parsing an XML document. The default value is 'false'. In the default case (<i>ERROR_HANDLER('false')</i>), if a parsing error occurs, the function aborts and throws an exception. If you enable error handling (<i>ERROR_HANDLER('true')</i>), if an error occurs while parsing a row, the function skips that row and goes to the next row. When the function completes the parsing, it outputs the nodes in the XML document that were error free. When you enable error handling, you can also specify that additional information be outputted about the errors. For example, if you use this clause: <code>ERROR_HANDLER('true;error_info:column1, column2')</code> the function adds the column "error_info" to the output table and adds to it the content of the input columns "column1" and "column2," separated by a semicolon. If you enable error handling without specifying an output column name, for example, <i>ERROR_HANDLER('true;column1, column2...')</i> , the default column name is "error_handler."
<i>ACCUMULATE</i>	Optional	Specifies a list of columns of the input to include in the output. Note that no column name can be the same as any column name indicated in the output schemas shown below.

Input Data

The input data is a table which contains at least one column with XML data to be flattened.

Malformed Documents

If malformed XML documents are encountered, the function skips them and continue to parse the next document.

Output

The output schema is chosen using the *OUTPUT* argument. There are three schemas from which to choose for the output:

Table 12 - 323: Output Schema Options

Schema	Description
fulldata	<p>Uses a schema with the following properties:</p> <ul style="list-style-type: none"> For each XML document, a row is emitted for each node with a depth lower than was specified using the <i>MAXDEPTH</i> argument. The 'out_nodeid' column contains the node's unique identifier within the document. D*Element is the node name. D*Attribute is "attributename=value[delimiter...]". D*Value is the text content of the node. D*ID is the 'out_nodeid' of D*Element. Generally, if the emitted node is in Depth N, elements from 'D0Element' to 'D(N-1)ID' will provide the information for its ancestors, and 'DNElement' to 'DNID' will be the information for the node itself. <p>The output schema 'fulldata' is:</p> <pre>TABLE output_table (<docid_columns>, -- columns specified in DocIDColumn argument out_nodeid int non null, -- the id of each node extracted, unique within each XML document D0Element varchar, -- root is in depth 0 D0Attributes varchar, D0Value varchar, D0ID int, D1Element varchar, D1Attributes varchar, D1Value varchar, D1ID int, D2Element varchar, D2Attributes varchar, D2Value varchar, D2ID int, D3Element varchar, D3Attributes varchar, D3Value varchar, D3ID int, D4Element varchar, D4Attributes varchar, D4Value varchar, D4ID int, D5Element varchar, D5Attributes varchar, D5Value varchar, D5ID int, <column_names>, -- other columns specified in ACCUMULATE argument);</pre>

Table 12 - 323: Output Schema Options

Schema	Description
parentchild	<p>Uses a schema that includes a column 'ParentID', which is the 'out_nodeid' of the element's parent.</p> <p>The output schema 'parentchild' is:</p> <pre>TABLE output_table (<docid_columns>, out_nodeid int non null, Element varchar non null, Attributes varchar, Value varchar, ParentID int, <column_names>, -- other columns specified in ACCUMULATE argument);</pre>
fullpath	<p>Uses a schema that can be described as halfway between the 'fulldata' and 'parentchild' schemas.</p> <p>The output schema 'fullpath' is:</p> <pre>TABLE output_table (<docid_columns>, out_nodeid int non null, Element varchar non null, Attributes varchar, Value varchar, D0ID int, D1ID int, D2ID int, D3ID int, D4ID int, D5ID int, <column_names>, -- other columns specified in the ACCUMULATE clause);</pre>

Examples

The examples in this section demonstrate:

- Example 1: Using different output schemas
- Example 2: Changing the *ATTRIBUTETONODE* setting

Example Input Data

Both of the examples that follow use the `xml_inputs` table.

Table 12 - 324: Example input table: `xml_inputs`

xid	xmldocument
1	<Book bookId="100"> <Authors> <Author authorId="1"> <Name>Ann Zhang</Name> </Author> </Authors> <Chapters> <Chapter chapterId="1"> <Abstract>ABC</Abstract> </Chapter> </Chapters> </Book>

Example 1: XMLRelation with different output schemas

This example uses the three different output schemas to flatten the XML document:

fulldata SQL-MR call

```
SELECT * FROM XMLRelation (
  ON xml_inputs
  TEXTCOLUMN('xmldocument')
  DocIDColumns('xid')
  MaxDepth('3')
  Output('fulldata')
);
```

fulldata Output

Table 12 - 325: Example output table: `fulldata`

xid	out_nodeid	D0 Element	D0Attributes	D0 Value	D0ID	D1 Element	D1 Attributes	D1Value	D1ID	D2 Element	D2 Attributes	D2 Value	D2ID	D3 Element	D3 Attributes	D3Value	D3ID
1	1	Book	bookId=100		1												
1	2	Book	bookId=100		1	Authors											
1	3	Book	bookId=100		1	Authors				Author	authorId=1						
1	4	Book	bookId=100		1	Authors				Author	authorId=1		3	Name	Ann	Zhang	4
1	5	Book	bookId=100		1	Chapters											
1	6	Book	bookId=100		1	Chapters				Chapter	chapterId=1		6				
1	7	Book	bookId=100		1	Chapters				Chapter	chapterId=1		6	Abstract	ABC		7

Sample SQL-MR call (parentchild)

```
SELECT * FROM XMLRelation (
    ON xml_relation_data
    TEXTCOLUMN('xmldocument')
    DocIDColumns('xid')
    MaxDepth('3')
    Output('parentchild')
)
ORDER BY out_nodeid;
```

Output

Table 12 - 326: Example output table: parentchild

xid	out_nodeid	Element	Attributes	Value	ParentID
1	1	Book	bookId=100		null
1	2	Authors			1
1	3	Author	authorId=1		2
1	4	Name		Ann Zhang	3
1	5	Chapters			1
1	6	Chapter	chapterId=1		5
1	7	Abstract		ABC	6

Sample SQL-MR call (fullpath)

Note that fullpath is the default Output setting, and as such, does not need to be specified explicitly, as is shown here.

```
SELECT * FROM XMLRelation (
    ON xml_relation_data
    TEXTCOLUMN('xmldocument')
    DocIDColumns('xid')
    MaxDepth('3')
    Output('fullpath')
)
ORDER BY out_nodeid;
```

Output

Table 12 - 327: Example output table: fullpath

xid	out_nodeid	Element	Attributes	Value	D0ID	D1ID	D2ID	D3ID
1	1	Book	bookId=100		1			
1	2	Authors			1	2		
1	3	Author	authorId=1		1	2	3	
1	4	Name		Ann Zhang	1	2	3	4
1	5	Chapters			1	5		

Table 12 - 327: Example output table: fullpath (continued)

xid	out_nodeid	Element	Attributes	Value	D0ID	D1ID	D2ID	D3ID
1	6	Chapter	chapterId=1		1	5	6	
1	7	Abstract		ABC	1	5	6	7

Example 2: XMLRelation with AttributeToNode

This example shows the same output as the fullpath example above, but with ATTRIBUTETONODE set to 'true'.

Example SQL-MapReduce call

```
SELECT * FROM XMLRelation (
    ON xml_inputs
    TEXTCOLUMN('xmldocument')
    DocIDColumns('xid')
    MaxDepth('3')
    ATTRIBUTETONODE('true')
)
ORDER BY out_nodeid;
```

Example 2 Output

Note that '~' in the output indicates that the corresponding element is an attribute in the XML document. If not shown, the corresponding element is a node.

Table 12 - 328: Example output table

xid	out_nodeid	Element	Attributes	Value	D0ID	D1ID	D2ID	D3ID
1	1	Book	bookId=100		1			
1	2	bookId	~	100	1	2		
1	3	Authors			1	3		
1	4	Author	authorId=1		1	3	4	
1	5	authorId	~	1	1	3	4	5
1	6	Name		Ann Zhang	1	3	4	6
1	7	Chapters			1	7		
1	8	Chapter	chapterId=1		1	7	8	
1	9	chapterId	~	1	1	7	8	9
1	10	Abstract		ABC	1	7	8	10

Example 3

Example Input

Table 12 - 329: Example input table: xml_inputs_error

xid	xmldocument
1	<Book bookId="100"></Book>
2	</Book>

Example SQL-MapReduce call

In this example, error handling is enabled.

```
SELECT * FROM XMLRelation (
    ON xml_inputs_error
    TEXTCOLUMN('xmldocument')
    DocIDColumns('xid')
    MaxDepth('1')
    ERROR_HANDLER('true;xmldocument')
) ;
```

Example output

Table 12 - 330: Example output table

xid	out_nodeid	Element	Attributes	Value	D0ID	D1ID	ERROR_HANDLER
1	1	Book	bookId=100		1		
2				</Book>;			

Error Messages

The function reads the full document into memory buffer before operating on it. One assumption is that none of the documents will exceed the available memory on the machine. If there are any empty documents, the function will ignore them and continue to process the next document.

You may see the following error message:

- ERROR: “SQL-MR function XMLRELATION requires column: ***”
REASON: Needed columns are missing from the relation named in the ON clause.
- ERROR: SQL-MR function XMLRELATION failed: The column '***' already exists in the output column list.
REASON: Column(s) in the 'DocIdColumns' and 'Accumulate' clause have a disallowed column name.
- ERROR: SQL-MR function XMLRELATION failed: The MAXDEPTH should be a positive integer.
REASON: The value of 'MaxDepth' is not a positive integer.
- ERROR: SQL-MR function XMLRELATION requires argument clause: ***
REASON: The needed arguments 'TEXTCOLUMN' or 'DocIDColumns' are missing.

- ERROR: SQL-MR function XMLRELATION failed: The OUTPUT should be 'fulldata', 'parentchild', or 'fullpath'.
REASON: The value of 'Output' is not 'fullpath', 'parentchild' or 'fullpath'.
- ERROR: SQL-MR function XMLRELATION failed: The format of 'ExcludeElements' is incorrect.
REASON: The format of string 'ExcludeElements' is incorrect.
- ERROR: SQL-MR function XMLRELATION failed: The depth of 'ExcludeElements' should satisfy MAXDEPTH.
REASON: The depth defined in 'ExcludeElements' is larger than 'MaxDepth'.

JSONParser

Summary

The JSONParser function is a tool used to extract the element name and text from JSON strings and output them into a flattened relational table.



Background

On the Internet, most data is exchanged and processed using JSON or XML, and then displayed using HTML. These languages have a great deal in common: they are based on named structures that can be nested, using a Unicode-based representation that is both human-readable and machine-readable.

Each language is optimized for its main function:

- HTML for representing web pages
- XML for representing document data
- JSON for representing programming language structures

In many applications, programmers work with only one of these formats. In others, programmers work with all three. For traditional programming structures, programming with JSON is significantly easier than programming with XML.

XQuery is the standard query language for XML, and has been implemented in databases, streaming processors, data integration platforms, application integration platforms, XML message routing software, web browser plugins, and other environments. But there is currently no standard query language for JSON. To solve this challenge, Teradata Aster has created the JSONParser SQL-MR function, which takes the JSON string as input and parses it as specified by the function arguments, outputting it as a relational table which can then be queried using SQL.

Usage

Permissions

You must grant EXECUTE on the function “JSONParser” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 1.3)

```
SELECT * FROM JSONParser
(
    ON tablename
    TEXT_COLUMN ('text_columnname')
    NODES ('parentnode/childnode'[,...])
    [SEARCH_PATH('nodename/nodename/...')]
    [DELIMITER('delimiter_string')]
    [MAX_ITEM_NUMBER('number')]
    [NODEID_OUTPUTCOLUMN_NAME('columnname')]
    [PARENTNODE_OUTPUTCOLUMN_NAME('columnname')]
    [ACCUMULATE('columnname'[,...])]
    [ERROR_HANDLER('true|false
        [;output_col_name :] input_col_name1,
        input_col_name2, input_col_name3,...)]]
)
;
```

Arguments

<i>TEXT_COLUMN</i>	Required	Specifies the column name from the input table which contains the JSON string.
<i>NODES</i>	Required	Specifies the parent/children pair. Should contain at least one parent/child pair, and all pairs specified must be in the same format. Multiple children can be specified as parent/{child1,child2,...}.
<i>SEARCH_PATH</i>	Optional	Specifies the path to find the direct value of the child. To reach the parent of the parent, include the parent of the parent in this path. When a path to the parent of the parent is supplied, all the siblings of the parent can be printed by including them in the NODES argument. If anything from root is to be parsed, then supply this argument as '/' (or leave it as an empty string).
<i>DELIMITER</i>	Optional	Specifies the delimiter used to separate multiple child values with the same name and which have the same parent node in the JSON String. If not defined, defaults to comma ','.
<i>MAX_ITEM_NUMBER</i>	Optional	The maximum number of nodes with the same name that will be displayed in the output.
<i>NODEID_OUTPUTCOLUMN_NAME</i>	Optional	The name of the column to use in the result schema to contain the identifier (from the input table) of the each node extracted. If not defined, defaults to 'out_nodeid'.
<i>PARENTNODE_OUTPUTCOLUMN_NAME</i>	Optional	The name of column to use in the result schema to contain the tag name of the parent node extracted. If not defined, defaults to 'out_parent_node'.
<i>ACCUMULATE</i>	Optional	The list of all the columns from the input table to include in the output.

<i>ERROR_HANDLER</i>	Optional	<p>Specifies how the function acts when it encounters a data problem. If not specified, the function aborts if the input table contains bad data (for example, invalid UTF-8 characters).</p> <p><i>ERROR_HANDLER</i> lets you specify an “additional” column to hold any rows that were rejected as having bad data, also referred to as the output column, in the output table. The log information in the additional column lets you easily identify which input table row contains unexpected data.</p> <p>There are two parameters you can pass to <i>ERROR_HANDLER</i>:</p> <ul style="list-style-type: none"> The first parameter tells the function whether to continue processing if bad data is encountered. ‘true’ means continue the processing without aborting. ‘false’ means abort the process when an error occurs. The second group of parameters designates the output and input columns. The parameters in this group, <i>output_col_name</i>: <i>input_col_name1</i>, <i>input_col_name2</i>, <i>input_col_name3</i>,... are optional. If you specify an output column, it will be added to the output, and bad rows are logged there. If you do not specify <i>output_col_name</i>, the function uses “<i>ERROR_HANDLER</i>” as the name of the output column. The error output column includes the data from the input columns specified using <i>input_col_name</i>x, when an error occurs. The data inserted into the output column will be merged from input columns and delimited by column using a semicolon. <p>Using <i>ERROR_HANDLER</i>(‘true’) without specifying input columns does not add any data to the output column.</p>
----------------------	----------	---

Input Data

The table used as input must contain a column with JSON data.

Output

A row is output for each node in the JSON string which has its name indicated as a parent node in the NODES argument.

The output table contains columns with the node ID, the parent node name, and the children nodes. The output also contains all columns specified in the ACCUMULATE argument.

- Arrays can be formatted as one of two types in JSON:
 - parent:[key:value,key:value]
For this type, use ‘parent/key’ in the NODES argument
 - parent[value,value]
For this type, use ‘parent/parent’ in the NODES argument
- Root sometimes has a key:value pair like that shown in example 2 below.
To get the value of such a pair, supply ‘/key’ for the NODES argument.

Examples

Example 1 Input Data

Table 12 - 331: Example input table: json_parser_data

id	data
1	{"menu": { "id": "1", "value": "File", "popup": { "menuitem": [{"value": "New", "onclick": "CreateNewDoc()"}, {"value": "Open", "onclick": "OpenDoc()"}, {"value": "Close", "onclick": "CloseDoc()"}] } }}

Example 1 SQL-MapReduce call

```
SELECT * FROM JSONParser (
    ON json_parser_data
    TEXT_COLUMN('data')
    NODES ('menu/{id,value}', 'menuitem/value')
    DELIMITER ('|')
    NODEID_OUTPUTCOLUMN_NAME ('ID')
    PARENTNODE_OUTPUTCOLUMN_NAME ('ParentName')
    ACCUMULATE ('id')
) ;
```

Example 1 Output

Table 12 - 332: Example output table

id	ID	ParentName	menu:id	menu:value	menuitem:value
1	1	menu	1	File	
1	2	menuitem			New Open Close

Example 2 Input Data

Table 12 - 333: Example input table: json_parser_data_2

id	data
1	{ "email": "xyz@asterdata.com", "glossary": { "title": "example glossary", "GlossDiv": { "title": "S", "GlossList": { "GlossEntry": { "ID": "SGML", "SortAs": "SGML", "GlossTerm": "Standard Generalized Markup Language", "Acronym": "SGML", "Abbrev": "ISO 8879:1986", "GlossDef": { "para": "A meta-markup language", "GlossSeeAlso": ["GML", "XML"] }, "GlossSee": "markup" } } } } }

Example 2 SQL-MapReduce call

```
SELECT * FROM JSONParser (
    ON json_parser_data_2
    TEXT_COLUMN('data')
    NODES('glossary/title','GlossDiv/title','GlossEntry/Abbrev',
    'GlossSeeAlso/GlossSeeAlso','/email')
    DELIMITER(' | ')
    ACCUMULATE('id')
    MAX_ITEM_NUMBER(10)
) ;
```

Example 2 Output

Table 12 - 334: Example output table

id	out_nodeid	out_parent_node	glossary:title	GlossDiv:title	GlossEntry:Abbrev	GlossSeeAlso:GlossSeeAlso	:email
1	1	glossary	example glossary				
1	2	GlossDiv		S			
1	3	GlossEntry			ISO 8879:1986		
1	4	GlossSeeAlso				GML XML	
1	5						xyz@aster data.com

Example 3: Parsing with Ancestor

These examples show two different ways parsing the same input table using the JSON ancestor.

```
SELECT * FROM JSONParser (
    ON json_parser_data_2
    TEXT_COLUMN('data')
    NODES('GlossEntry/ID')
    SEARCH_PATH('/glossary/GlossDiv/GlossList')
    DELIMITER(' | ')
    ACCUMULATE('id')
    MAX_ITEM_NUMBER(10)
) ;
```

Table 12 - 335: Example output table

id	out_nodeid	out_parent_node	GlossEntry:ID
1	1	GlossEntry	SGML

```
SELECT * FROM JSONParser (
    ON json_parser_data_2
    TEXT_COLUMN('data')
    NODES('/email')
    SEARCH_PATH('/')
    DELIMITER(' | ')
    ACCUMULATE('id')
    MAX_ITEM_NUMBER(10)
) ;
```

Table 12 - 336: Example output table

id	out_nodeid	out_parent_node	:email
1	1		xyz@asterdata.com

Example 4: Specifying ERROR_HANDLER when Calling JSONParser

Here is an example showing the use of the *ERROR_HANDLER* clause:

```
SELECT * FROM JSONParser (
    ON json_parser_data
    TEXT_COLUMN('data')
    NODES('menuitem/value')
    ERROR_HANDLER('true; bad_row : id1, id2')
) ORDER BY 1;
```

Apache Log Parser

Summary

This function parses Apache log file content to a schema from a given server access log and extracts multiple columns of structural information, including search engines and search terms. Custom log formats are supported, but if not specified, the default format is the NCSA extended/combined log format.



Background

The `apache_log_parser` function parses Apache log files with the following constraints:

- Log files are loaded into a table
- One line of the Apache log file is loaded to one row in the table
- The content must conform to the supplied log format if a custom log format string is given.

- If no custom format string is given via the LOG_FORMAT argument, the function will parse the logs against NCSA extended/combined format defined as:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

Background on Apache log configuration

The Apache web server enables users to customize server access logs in terms of both the content and the reporting conditions. For example, the server may be configured to report remote host name, time of each request, and the requested path only when the page fails to open.

The desired log format must be present in the Apache server configuration file, as an argument of these directives:

- *LogFormat*: is used to define a custom format only.
- *CustomLog*: is used to define a log file under certain custom formats in one step.

The characteristics of the request itself are logged by placing "%" directives in the format string. At logging time, the directives are substituted with specific values.

Below is an example from a configuration file. Please see http://httpd.apache.org/docs/current/mod/mod_log_config.html for more details.

```
#  
# The following directives define some format nicknames for use with  
# a CustomLog directive (see below).  
# If you are behind a reverse proxy, you might want to change %h into  
#{X-Forwarded-For}i  
#  
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-  
Agent}i\" vhost_combined  
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\""  
combined  
LogFormat "%h %l %u %t \"%r\" %>s %O" common  
LogFormat "%{Referer}i -> %U" referrer  
LogFormat "%{User-agent}i" agent
```

Parsing

Apache Log Parser Item-Name Mapping

Table 12 - 337: Apache Log Parser Item-Name Mapping

log item	column name	output examples
%h	remote_host	153.65.52.112
%a	remote_IP	153.65.52.112
%A	local_IP	153.65.52.112
%t	request_time	[22/Jun/2012:17:27:02 -0700]
%b	bytes_sent_CLF	512
%B	bytes_sent	455
%O	bytes_sent_including_header	512

Table 12 - 337: Apache Log Parser Item-Name Mapping (continued)

log item	column name	output examples
%I	bytes_received_including_header	512
%p	canonical_server_port	80
%{canonical}p	canonical_server_port	80
%{local}p	actual_server_port	80
%{remote}p	actual_client_port	7777
%P	process_ID	8311
%k	live_connections	2
%D	request_duration_microseconds	312
%T	request_duration_seconds	0
%U	requested_URL	/index.html
%l	remote_log_name	
%u	remote_user	
%f	requested_file	
%{VARNAME}i	request:VARNAME	
%{Referer}i	Referer	
%{VARNAME}o	reply:VARNAME	
%{VARNAME}n	note:VARNAME	
%{VARNAME}e	env:VARNAME	
%{VARNAME}C	cookie:VARNAME	
%V	server_name	
%v	canonical_server_name	
%L	log_ID	
%H	protocol	HTTP/1.1
%m	method	GET
%q	query	
%X	connection_status	X = Connection aborted before the response completed. + = Connection may be kept alive after the response is sent. - = Connection will be closed after the response is sent.

Table 12 - 337: Apache Log Parser Item-Name Mapping (continued)

log item	column name	output examples
%r	request_line	
%>s	final_status	404
%<s	original_request_status	
%s	original_request_status	
%R	handler	

Usage

Permissions

You must grant EXECUTE on the function “apache_log_parser” to the database user who will run the function.

Before running this function, you must obtain the right to run it using the GRANT EXECUTE command. For more information, see [“Set Permissions to Allow Users to Run Functions” on page 53](#).

Syntax (version 2.1)

```
SELECT *
FROM apache_log_parser
  (ON {table_name|view_name|(query)}
   LOG_COLUMN('log_column_name')
   [LOG_FORMAT('format_string')]
   [EXCLUDE_FILES('.file_suffix',['.file_suffix2',...])]
   [RETURN_SEARCH_INFO('true'|'false')])
);
```

Arguments

<i>LOG_COLUMN</i>	Required	Name of the column whose contents will be parsed. Only one column is permitted.
<i>LOG_FORMAT</i>	Optional	The Apache Log Format string is used to generate the server access logs. This format string can be found in the apache server configuration file. The default log format is NCSA extended/combined format. Log items in the format string will each have a corresponding output column built with the column names listed in the Example Input Apache Log Parser Item-Name Mapping table .
<i>EXCLUDE_FILES</i>	Optional	Comma separated file suffixes to exclude. All the files to be excluded should be specified in the EXCLUDE_FILES clause separately, within single quotes. The default is '.png', '.xml', '.js'.
<i>RETURN_SEARCH_INFO</i>	Optional	A true or false value that specifies whether to return search information. If 'true', the search engine and the search terms, if existing, are extracted into two output columns. The default is 'false'. Three search engines are supported for identifying search activities: Google, Bing, and Yahoo. More complete parsing capabilities are provided for Google.

Output Schema

The function emits a row for the log content of each row it parses, unless the requested page is one of the *EXCLUDE_FILES* types ('.png', '.xml', or '.js') which does not emit any output for that input row.

The output schema is not fixed, but is subject to the supplied custom log format and the clause arguments on each function invocation. A typical output schema may contain the following columns:

Table 12 - 338: Output schema columns

Column	Description
<i>remote_host</i>	The remote host that made the HTTP request.
<i>request_time</i>	The timestamp when the HTTP request is made.
<i>requested_page</i>	The landing page.
<i>final_status</i>	Identifies the status for requests that have been internally redirected. Here, the final request should be consulted, not the original request.
<i>bytes_sent_CLF</i>	The response size in bytes in Custom Log Format (CLF).
<i>request_{User-agent}</i>	The referring URL from which the visitor has arrived.

The following two columns are extracted only when RETURN_SEARCH_INFO = 'true' and the access logs contain referrer information:

Table 12 - 339: Columns extracted when RETURN_SEARCH_INFO = 'true'

Column	Description
<i>search_engine</i>	Google, Bing, and Yahoo. If there is no search engine referral, the output is blank.
<i>search_terms</i>	the search terms entered by the search engine user, which led to landing on the page.

Example Input Apache Log Parser Item-Name Mapping table

Table 12 - 340: Example output table: web_log_ut

id	data
1	168.187.7.114 - - [27/Mar/2011:00:16:49 -0700] "GET / HTTP/1.0" 200 7203 "http://search.yahoo.com/search;_ylt=AtMGk4Fg.FlhWyX_ro.u0VybvZx4?p=ASTER&toggle=1&cop=mss&ei=UTF-8&fr=yfp-t-383-1" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.2)"
2	75.36.209.106 - - [20/May/2008:15:43:57 -0400] "GET / HTTP/1.1" 200 15251 "http://www.google.com/search?hl=en&q=%22Aster+Data+Systems%22" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; YPC 3.2.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; MS-RTC LM 8)"

Table 12 - 340: Example output table: web_log_ut (continued)

id	data
3	159.41.1.23 - - [06/Jul/2010:07:19:45 -0400] "GET /public/js/common.js HTTP/1.1" 200 16711 "http://www.wooloo.org/wonjoo" "Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"
4	llf520029.crawl.yahoo.net - - [29/May/2008:23:15:15 -0400] "GET /resources/images/support HTTP/1.0" 301 187 "" "Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)"

Example SQL-MapReduce Call

```
SELECT remote_host, requested_page,
       final_status, search_engine, search_terms
  FROM apache_log_parser
  (
    ON web_log_ut
    LOG_COLUMN('data')
    RETURN_SEARCH_INFO('true')
  ) ORDER BY remote_host;
```

Example Output from Apache Log Parser

Table 12 - 341: Example output table

remote_host	request_time	requested_page	final_status	search_engine	search_terms
168.187.7.114	2008-05-20 15:43:57.0	/	200	yahoo	ASTER
75.36.209.106	2011-03-27 11:45:47.0	/	200	google	"Aster Data Systems"
llf520029.crawl.yahoo.net	2008-05-29 23:15:15.0	/resources/images/support	301		

Errors

Error Messaging

You may see the following error messages:

- Requires input rows to have a column of specified name: <log_column_name>
- Requires the specified log column <log_column_name> to be of NativeType String.

outlierFilter

Summary

This function removes outliers from their data set. The input data is expected to have as many as millions of variables listed as 'attribute_value' pairs. This function provides several different methods for defining outliers: percentile, Tukey's method, and Carling's modification.



Usage

Syntax (version 1.1)

```

SELECT * FROM outlierFilter (
    ON (SELECT 1)
    PARTITION BY 1
    inputTable('input_table_name')
    outputTable('output_table_name')
    filterCol('data_col')
    [groupByCols('group_by_col1', 'group_by_col2', ...)]
    [method({'percentile'} | 'tukey' | 'carling' | 'MAD-median')]
    [useApproxPercentile('false' | 'true')]
    [percentileValues('perc_lower', 'perc_upper')]
    [percentileTolerance ('accuracy')]
    [IQRmultiplier ('k')]
    [removeTail('upper' | 'lower' | {'both'})]
    [replacementValue('newval')]
    [domain('ip_address')]
    [database('db_name')]
    [userid('user_id')]
    password('password')
);
  
```

Arguments

<i>INPUTTABLE</i>	Required	Input table contains the numeric data to be filtered, along with list of groupby columns.
<i>OUTPUTTABLE</i>	Required	Output table is the table where output is stored. The output table copies the structure of the input table exactly (including the partition-by column) but with outliers removed.
<i>FILTERCOL</i>	Required	Name of the numeric column to be filtered for outliers.
<i>GROUPBYCOLUMNS</i>	Optional	List of columns to group the data by. If data schema is in Name:Value format, then Name should be included in GROUPBYCOLUMNS.
<i>METHOD</i>	Optional	Method of outlier identification. Default value is 'percentile'

<i>USEAPPROXPERCENTILE</i>	Optional	The percentiles used as filter limits can be calculated exactly (false) or approximately (true). Approximate percentiles will typically be faster—but this method may fail when the number of groups is large (>1M). Try both and see which is faster for your data set. Default is 'false'. NOTE: Only the boolean values true or false can be used. Other values like 0/1 or yes/no do not produce the expected result.
<i>PERCENTILEVALUES</i>	Optional	percentile values used only for method('percentile') filtering. Can be one or two numbers. If a single number 'a' is provided, the filter limits will be interpreted as [a, 100-a]. If two values ('a','b') are provided then the limits will be [a,b]. Default value is 5, 95.
<i>PERCENTILETOLERANCE</i>	Optional	Accuracy of percentiles used for filtering. Default is 0.5%
<i>IQRMULTIPLIER</i>	Optional	Multiplier of interquartile range used in Tukey filtering. Default is 1.5
<i>REMOVETAIL</i>	Optional	The side of the distribution to be filtered. Default is 'both' (options are 'upper' and 'lower')
<i>REPLACEMENTVALUE</i>	Optional	Values to replace outliers with. Default is 'delete' which removes the entire row from the output table. Other options include 'NULL' in which case outliers will be replaced by nulls, 'median' which replaces outliers with the median value for that group, or any numeric value such as '1.2' or '0'.
<i>DOMAIN</i>	Optional	IP address of the queen node. Default domain is queen of the current cluster.
<i>DATABASE</i>	Optional	This is the name of the database where the input table is present. Default database is beehive.
<i>USERID</i>	Optional	The Aster Database user name of the user. Default userid is beehive.
<i>PASSWORD</i>	Required	The Aster Database password of the user.

Input

The target variable must be present in a single column. In the example below, the values in blue are outliers and should be removed. The Value columns must be of numeric type.

Table 12 - 342: Sample input table

RowID	GroupCol	Variable_Name	Value
1	A	Pressure	500
2	A	Pressure	99999
3	A	Temperature	23
4	A	Temperature	80
5	A	Temperature	100
6	A	Temperature	90
7	A	Pressure	-10

SQL-MapReduce Call

```
SELECT * FROM outlierFilter(
    ON (SELECT 1)
    PARTITION BY 1
```

```

    inputTable('input_table')
    outputTable('output_table')
    filterCol('value')
    method('percentile')
    percentileValues('1','90')
    removeTail('both')
    replacementValue('null')
    groupByCols('groupcol', 'Variable_Name')
    userid('beehive')
    password('beehive')
) ;

```

Output

The function creates a table by the name of “output_table” in the same schema as the input table. The output schema is identical to the input schema but with the outlier values replaced by a chosen value (or the rows deleted). Any extra columns in the input data (including additional numeric data columns) will be replicated in the output schema.

Table 12 - 343: Sample Output table

RowID	GroupCol	Variable_Name	Value
1	A	Pressure	500
2	A	Pressure	NULL
3	A	Temperature	23
4	A	Temperature	80
5	A	Temperature	NULL
6	A	Temperature	90
7	A	Pressure	-10

Limitations

Currently, MAD-median filtering is not supported.

IpGeo

Summary

IpGeo lets you map IP addresses to location information. You can use this information to identify the geographical location of a visitor. This information includes country, region, city, latitude, longitude, ZIP code, and ISP.



Background

Using IpGeo, you can improve the effectiveness of online applications including: targeted online advertising, content localization, geographic rights management, enhanced analytics, and online security and fraud prevention.

Usage

Syntax (version 1.0)

```

IpGeo(
    ON {table_name|view_name|(query)}
    IPADDRESSCOLUMN('ip_address_column_name')
    CONVERTER('file_name','class_name')
    [IPDATABASELOCATION('location_of_the_geolocation_database')]
    [ACCUMULATE('column_name_list')]
)
  
```

<code>IPADDRESSCOLUMN</code>	Required	Name of the column whose content is a IP address. Only one column is permitted.
<code>CONVERTER</code>	Required	The JAR filename and the name of the class that converts the IP address to location information. The JAR file should be installed on the Aster Database and the class name should be the full name, with includes the package information. The <code>file_name</code> and <code>class_name</code> parameters are case sensitive. To create a new class, see “ Extending IpGeo ” on page 516.
<code>IPDATABASELOCATION</code>	Optional	The location of the IP database that matches IP addresses to locations. The IP databases can be stored in the file system or in Aster Database. If the data is stored in a file system, each worker should have the same path, and the absolutized path should be set in this parameter. If the data is installed in Aster Database, this argument is ignored.
<code>ACCUMULATE</code>	Optional	List of columns you want to return in the input table.

Input

An input table listing user IDs and their corresponding IP addresses.

Output

Table 12 - 344: Output schema

Column Name	Type	Description
country_code	varchar	Country code
country_name	varchar	Country name
state	varchar	State or region name
city	varchar	City name
postal_code	varchar	Postal code
latitude	decimal(6,4)	Latitude
longitude	decimal(7,4)	Longitude
isp	varchar	Name of the ISP
organization	varchar	Name of the organization that owns this IP address
organization_type	varchar	Organization type
area_code	int	Area code
metro_code	int	Metro code
dma_code	int	DMA code

Example

Example Input

Table 12 - 345: Input table: ipgeo_1

id	ip
1	10.1.1.27
1	153.65.16.10
1	10.0.0.20
1	202.106.0.20
1	202.106.
1	ddd

Example SQL-MapReduce call

This call specifies /home/maxmind/ as the location of the IP database in the file system.

```
SELECT * FROM IPGEO(ON ipgeo_1
    IpAddressColumn('ip')
    IpDatabaseLocation('/home/maxmind/')
    Converter('MaxMindLite.jar',
        'com.asterdata.sqlmr.analytics.location.ipgeo.MaxMindLite'))
```

```

        ACCUMULATE('id', 'ip')
);

```

This call assumes that the IP database is stored as a file in Aster Database.

```

SELECT *
FROM IPGEO(
    ON ipgeo_1
    IpAddressColumn('ip')
    Converter('MaxMindLite.jar',
              'com.asterdata.sqlmr.analytics.location.ipgeo.MaxMindLite')
    ACCUMULATE('id', 'ip')
);

```

Example Output

Table 12 - 346: Output table (columns 1–7)

id	ip	country_code	country_name	state	city	postal_code
1	10.1.1.27					
1	153.65.16.10	US	United States	Ohio	Miamisburg	45342
1	10.0.0.20					
1	202.106.0.20	CN	China		Beijing	
1	202.106.					
1	ddd					

Table 12 - 347: Output table (columns 1 and 8–15)

id	...	latitude	longitude	isp	organization	organization_type	area_code	metro_code	dma_code
1	...								
1	...	39.6182	-84.2488				937	542	542
1	...								
1	...	39.9289	116.3883				0	0	0
1	...								
1	...								

Extending IpGeo

Because IpGeo cannot cover all the IP database providers for technical and license reasons, you can extend this function to support new database providers.

To extend IpGeo:

- 1 Create a new class that implement the interface Converter. The interface Converter is defined as:

```

package com.asterdata.sqlmr.analytics.location.ipgeo;
public interface Converter
{
    /**
     */
}

```

```

        * initialize a Converter instance with corresponding resource
        * @param ipDatabasePath
        */
void initialize(String ipDatabasePath);

/**
 * release resources used by this instance before the SQL-MR
function close
*/
void finalize();

/**
 * Lookup location information for the input ipv4 address and write
the result to a IpLocation instance
 * @param ip
 *      input, IP address in ipv4 format
 * @param ipLocation
 *      output, to hold the location information
 */
void findIpv4(String ip, IpLocation ipLocation);

/**
 *
 * Lookup location information for the input ipv6 address and write
the result to a IpLocation instance
 * @param ip
 *      input, IP address in ipv6 format
 * @param ipLocation
 *      output, to hold the location information
 */
void findIpv6(String ip, IpLocation ipLocation);
}

```

Class `IpLocation` is designed to hold the location information and emit it. The code has get and set functions for the following member variables corresponding to the SQL-MR function output:

```

private String countryCode = null;
private String countryName = null;
private String state = null;
private String city = null;
private String postalCode = null;
private float latitude = -1;
private float longitude = -1;
private String isp = null;
private String organization = null;
private String organizationType = null;
private int areaCode = -1;
private int metroCode = -1;
private int dmaCode = -1;
}

```

The following implementation of MaxMindLite is an example converter for the maxmind lite database.

```
/*
 * Copyright (c) 2013 by Teradata Corporation. All rights reserved.
 * TERADATA CORPORATION CONFIDENTIAL AND TRADE SECRET
 *
 */
package com.asterdata.sqlmr.analytics.location.ipgeo;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import com.asterdata.ncluster.sqlmr.IllegalUsageException;
import com.asterdata.ncluster.sqlmr.data.InstalledFile;
import com.maxmind.geoip.Location;
import com.maxmind.geoip.LookupService;
import com.maxmind.geoip.regionName;
import com.asterdata.sqlmr.analytics.location.ipgeo.Converter;

/**
 * A Converter implementation for MaxMind Lite version
 */
public class MaxMindLite implements Converter
{
    private LookupService lookupService_ = null;
    private LookupService lookupServiceV6_ = null;

    private static final String CITY_DATABASE = "GeoLiteCity.dat";
    private static final String CITY_DATABASE_V6 = "GeoLiteCityv6.dat";

    private String tmpCityDatabase_ = null;
    private String tmpCityDatabaseV6_ = null;
    //initialize the lookup service
    public void initialize(String ipDatabasePath)
    {
        if(ipDatabasePath == null)
        {
            loadDefaultDatabase();
            lookupService_ = initializeService(tmpCityDatabase_);
            lookupServiceV6_ = initializeService(tmpCityDatabaseV6_);
        }
        else
        {
            String path = ipDatabasePath.endsWith("/") ?
                ipDatabasePath + CITY_DATABASE : ipDatabasePath + "/" + CITY_DATABASE;
            lookupService_ = initializeService(path);

            path = ipDatabasePath.endsWith("/") ?
                ipDatabasePath + CITY_DATABASE_V6 : ipDatabasePath + "/" + CITY_DATABASE_V6 ;
            lookupServiceV6_ = initializeService(path);
        }
    }

    //find address according to ipv4 address
    public void findIpv4(String ip, IpLocation ipLocation)
    {
        Location location = lookupService_.getLocation(ip);

        if (location != null) {
            setOutput(location, ipLocation);
        }
    }
}
```

```

//find address according to ipv6 address
public void findIpv6(String ip, IpLocation ipLocation)
{
    Location location = lookupServiceV6_.getLocationV6(ip);

    if (location != null) {
        setOutput(location, ipLocation);
    }
}

//release resources
public void finalize()
{
    if(lookupService_ != null)
    {
        lookupService_.close();
    }

    if(lookupServiceV6_ != null)
    {
        lookupServiceV6_.close();
    }

    if(tmpCityDatabase_ != null)
    {
        new File(tmpCityDatabase_).delete();
    }

    if(tmpCityDatabaseV6_ != null)
    {
        new File(tmpCityDatabaseV6_).delete();
    }
}

//Set the output to ipLocation from location, provided by Maxmind
private void setOutput(Location location, IpLocation ipLocation)
{
    ipLocation.setCountryCode(location.countryCode);
    ipLocation.setCountryName(location.countryName);

    if (location.countryCode != null && location.region != null) {
        ipLocation.setState( regionName.regionNameByCode(location.countryCode,
location.region));
    }

    ipLocation.setCity(location.city);
    ipLocation.setAreaCode(location.area_code);
    ipLocation.setDmaCode(location.dma_code);
    ipLocation.setLatitude(location.latitude);
    ipLocation.setLongitude(location.longitude);
    ipLocation.setMetroCode(location.metro_code);
    ipLocation.setPostalCode(location.postalCode);
}

//save the default IP database files to file system
private void loadDefaultDatabase()
{
    tmpCityDatabase_ = downloadFile(CITY_DATABASE);
    tmpCityDatabaseV6_ = downloadFile(CITY_DATABASE_V6);

}

//save a file installed in Aster to file system
private String downloadFile(String file)
{
    BufferedInputStream in = null;
    try

```

```

{
    in = new BufferedInputStream(InstalledFile.getFile(file).getStream());
}
catch (FileNotFoundException e1)
{
    throw new IllegalUsageException("CAN'T found the default IP database," +
        " please check whether following file has been installed to Aster: " + file);
}

String tmpFile = "/tmp/" + file + System.currentTimeMillis();
byte[] buffer = new byte[1024];
BufferedOutputStream out = null;

try
{
    out = new BufferedOutputStream(new FileOutputStream(tmpFile));
    for(int len = in.read(buffer); len>-1; len=in.read(buffer))
    {
        out.write(buffer, 0, len);
    }
}
catch (FileNotFoundException e)
{
    throw new IllegalUsageException("CAN'T create a tmp file: " + tmpFile
        + ". Please check the conflicts in the system");
}
catch (IOException e)
{
    throw new IllegalUsageException("CAN'T write to tmp file: " + tmpFile
        + ". Please check if folder /tmp has more than 100M bytes free space.");
}
finally
{
    try{
        if (in != null) in.close();
        if (out != null) out.close();
    }
    catch(IOException e)
    {
        //do nothing
    }
}

return tmpFile;
}

//Return a lookupService for the specified file
private LookupService initializeService(String file)
{
    if (new File(file).exists())
    {
        try
        {
            return new LookupService(file, LookupService.GEOIP_MEMORY_CACHE );
        }
        catch (IOException e)
        {
            throw new IllegalUsageException("CAN'T initialize LookupService. See details:"+
+ e.getMessage());
        }
    }
    else
    {
        throw new IllegalUsageException("CAN'T find IP database: " + file);
    }
}
}

```

- 2** Compile the new Converter class and package it with all the dependant libraries in a JAR file.
- 3** Install the JAR file on Aster Database.
- 4** When calling the IpGeo SQL-MR function, set the and JAR filename and class name parameters of the *CONVERTER* argument to the names of your JAR file and class.

CHAPTER 13 Visualization Functions

- [NpathViz](#)
- [CfilterViz](#)

NpathViz

Summary and Background

NpathViz is a SQL-MR function that visualizes the output of the Teradata Aster nPath SQL-MR function.

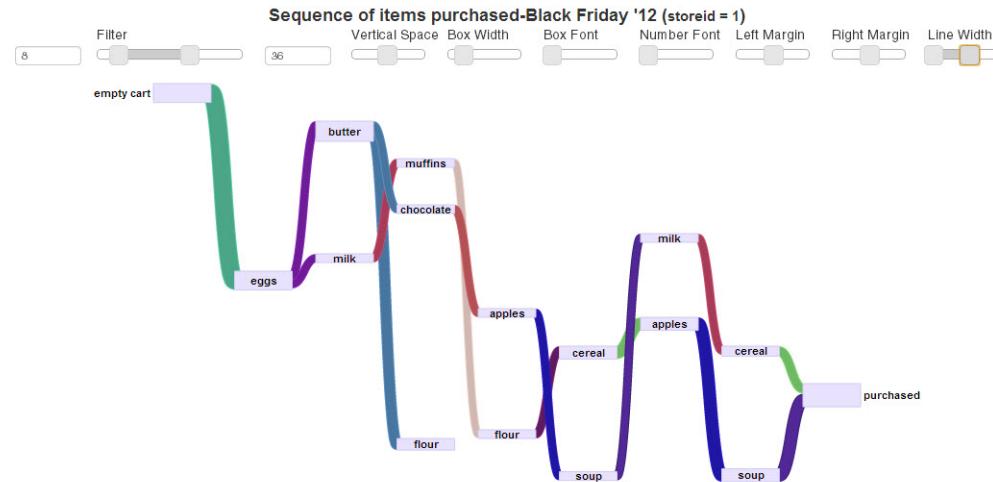


NpathViz generates these visualization types:

- [Sankey](#)
- [Tree](#)
- [Sigma](#)
- [Chord](#)
- [GEXF](#)
- [Graphviz](#)

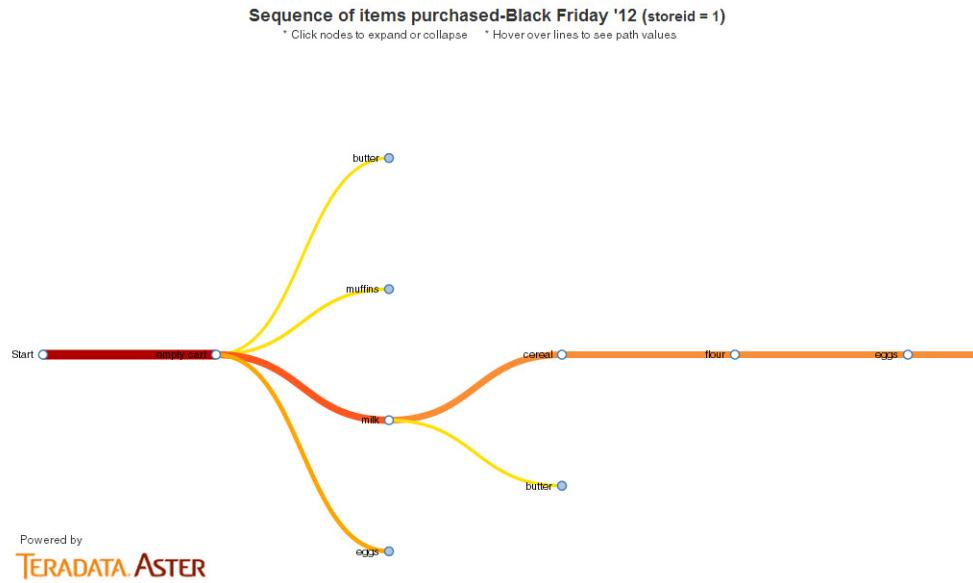
Sankey

Sankey diagrams are special types of flow diagrams that let you visualize flow quantity, which is represented by the width of the flow lines. You can view Sankey visualizations using Aster Lens.



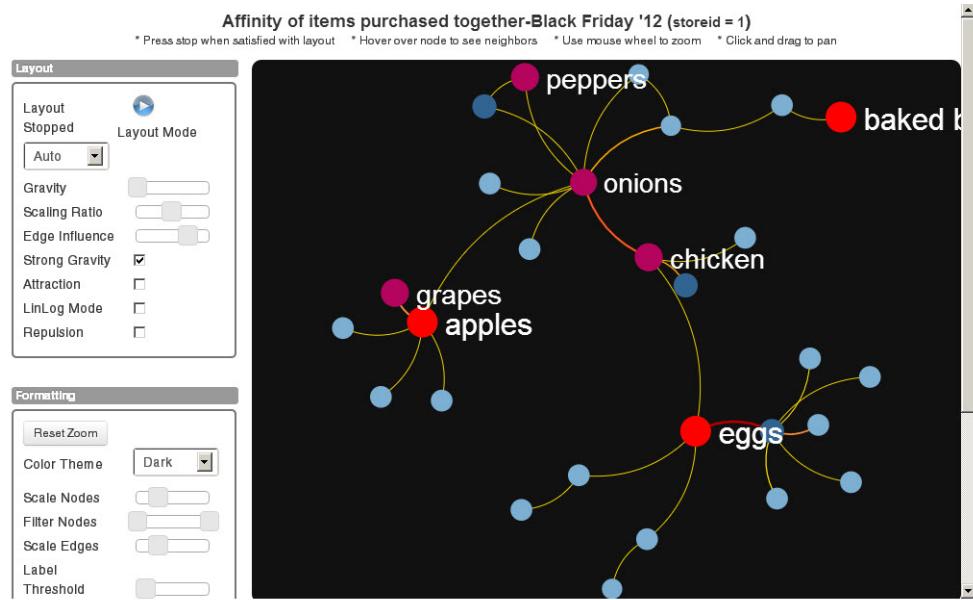
Tree

Tree diagrams let you visualize hierarchy and flow quantity. The width of the node connectors in the diagram represent the flow quantity. You can view Tree visualizations using Aster Lens.



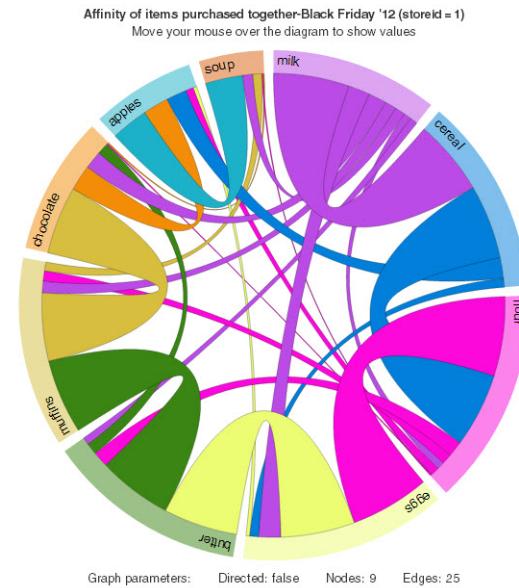
Sigma

Lets you generate graphs or network diagrams. A Sigma diagram shows you the vertices and edges of a graph. You can view Sigma visualizations using Aster Lens.



Chord

Chord diagrams let you explore relationships between entity groups. You can view Chord visualizations using Aster Lens.



Chord diagrams are best when the number of nodes is small, for example less than 50, and you want to show direction. The nodes are sized by the chord visualization engine to match the sum of the chord sizes. In this sense, the nodes in the chord and graph visualizations are sized in a similar fashion. Chord visualizations with more than 50 nodes might not display.

GEXF

You can choose to generate GEXF graphs. GEXF graphs are rendered using Gephi, an open-source network analysis and visualization software package. To view GEXF visualizations, download them using Aster Lens, then open them on your system.

Graphviz

You can choose to generate Graphviz graphs. Graphviz is an open-source visualization software. To view Graphviz visualizations, download them using Aster Lens, then open them on your system.

Usage

Syntax (version 1.0)

```
SELECT *
FROM NpathViz (
    ON <table|view|query> [AS 'input'] PARTITION BY i_col_name
    [ON <table|view|query> [AS 'aggregates'] PARTITION BY a_col_name]
    FREQUENCY_COL('score_col')
    PATH_COL('path_col')
    [GRAPH_TYPE('sankey'||'sigma'||'chord'||'tree'||'gexf'||'graphviz')]
    [ACCUMULATE('col_name')]
    [ARGUMENTS('key1=value1','key2=value2','key3=value3')]
    [JUSTIFY('left'||'right')]
    [DIRECTED('true'||'false')]
    TITLE('graph_title')
    [SUBTITLE('graph_subtitle')]
    [SANKEY_SORTING('true')]
    [SANKEY_NODE_ORDER(node_list)]
    [SANKEY_NODE_ORDER_ARRAY('colNumber=>nodeList',...)]
)
;
```

Arguments

ON	Required	The first ON clause specifies the input table. If you specify a second input, you must use 'input' as the alias for this clause. (Optional) The second ON clause specifies a user-defined aggregate. You must use 'aggregates' as the alias for this clause.
FREQUENCY_COL	Required	Specifies the name of the scoring column in the nPath output table. This column defines the affinity strength between the nodes (the score a path between nodes is associated with). The score is typically a count of how many times that path was followed.
PATH_COL	Required	Specifies the name of the column containing the path to the resulting graph.
GRAPH_TYPE	Optional	Specifies the graph type. Default: 'sankey'.
ACCUMULATE	Optional	Columns from the input table that you want in your output. Note: The ACCUMULATE argument can only contain the names of the columns that have been specified in the PARTITION BY clause. If not, the output is inconsistent.

<i>ARGUMENTS</i>	Optional	<p>The user-defined key-value pairs to be displayed in the output. The key/value pairs are case-sensitive. The function does not change their capitalization.</p> <p>Additionally, the function does not perform any trimming of the content. The characters to the left of the “=” operator define the column name. The characters to the right define the value inserted into the column. The value is repeated for every row generated by the function.</p> <p>Note: The single-quote, double-quote, and backslash characters are not allowed in keys. If you use any of these characters in a key, the function displays an error message.</p>
<i>JUSTIFY</i>	Optional	<p>(Applies only to Sankey and Tree diagrams) For tree diagrams, this argument specifies the flow direction ('left' or 'right'). For Sankey diagrams, this argument specifies element alignment.</p> <p>Default: 'left'.</p>
<i>DIRECTED</i>	Optional	<p>(Applies to SIGMA, CHORD, GEXF, and Graphviz diagrams) Specifies whether the graph is directed ('true') or undirected ('false').</p> <p>Default: 'false'.</p>
<i>TITLE</i>	Required	Specifies the title of the visualization diagram. Appears as a separate column in the output.
<i>SUBTITLE</i>	Optional	Specifies the subtitle of the visualization diagram. Appears as a separate column in the output.
<i>SANKEY_SORTING</i>	Optional	<p>(Sankey diagram only) Specifies whether the nodes in the Sankey diagram are sorted and displayed alphabetically ('true'). Default: 'false'.</p> <p>Sorting Sankey diagrams is useful when comparing them.</p>
<i>SANKEY_NODE_ORDER</i>	Optional	<p>(Applies only to Sankey diagram) Specifies a comma-separated, quoted list of Sankey node names. The Sankey nodes for all columns appear in the specified order. Nodes not specified in this argument are sorted only if SANKEY_SORTING is set to 'true'.</p>
<i>SANKEY_NODE_ORDER_ARRAY</i>	Optional	<p>(Applies only to Sankey diagram) Specifies a comma-separated list that lets you control the node order per Sankey column. Nodes not specified in the argument are sorted only if SANKEY_SORTING is set to 'true'.</p> <p>This argument provides you with granular control over node order in the diagram. Using this argument, you can specify a different node order per Sankey column.</p> <p>Each element in the list has this syntax:</p> <p><i>colNumber=>nodeList</i></p> <p>Where <i>colNumber</i> specifies the number of a Sankey column, with 1 representing the left-most column.</p>

Input

NpathViz takes as input one or two tables/relations:

- Output table generated by nPath.
- (Optional) A user-defined table containing aggregates.

Output

Depending on whether the PARTITION BY clause is used, the output of NpathViz is a table with one or more rows. Each row contains a ZIP archive stored in the blob column. This archive contains all of the files needed to display the visualized data. In addition, the function outputs visualization metadata, including user-defined metadata.

[Table 13 - 348](#) describes the default schema of the output table generated by NpathViz:

Table 13 - 348: Output table schema

Column	Type
id	bigint
blob	bytea
time_stamp	timestamp without time zone
graph_type	character varying
title	character varying

To add additional metadata columns, use the *ACCUMULATE* and *ARGUMENTS* clauses.

To view the visualizations generated by NpathViz, use [Aster Lens](#).

Examples

Example Input

The input used in this example is the output of this Teradata Aster nPath query:

```
SELECT * FROM npath (
    ON (select * from omniture_mobile_users_aggregated) AS omni
    PARTITION BY omni.visitid_low, omni.visitid_high, omni.visit_num
    ORDER BY omni.hit_time_gmt
    MODE(NONOVERLAPPING)
    PATTERN ('CHECKOUT_INTENT.(NOTHANKYOU){1,3}.OTHERNOTHANKYOU*$')
    SYMBOLS (
        pagename IN ('Cart : Billing and Shipping',
                     'Cart : Checkout : Process Order') AS CHECKOUT_INTENT,
        pagename NOT IN ('Cart : Checkout : Order Complete',
                         'InternationalCart : InternationalCheckout : Order Complete')
                     AS NOTHANKYOU,
        pagename NOT IN ('Cart : Checkout : Order Complete',
                         'InternationalCart : InternationalCheckout : Order Complete')
                     AS OTHERNOTHANKYOU)
    RESULT (
        FIRST(hit_time_gmt OF CHECKOUT_INTENT) AS hit_time_gmt,
        FIRST(ip OF CHECKOUT_INTENT) AS ip,
        FIRST(visid_low OF CHECKOUT_INTENT) AS visid_low,
        FIRST(visid_high OF CHECKOUT_INTENT) AS visid_high,
        FIRST(visit_num OF CHECKOUT_INTENT) AS visit_num,
        FIRST(geo_region OF CHECKOUT_INTENT) AS geo_region,
        FIRST(geo_country OF CHECKOUT_INTENT) AS geo_country,
        FIRST(1 OF ANY(CHECKOUT_INTENT, NOTHANKYOU)) AS cnt,
        ACCUMULATE(pagename OF ANY(CHECKOUT_INTENT, NOTHANKYOU)) AS path));

```

The output of this query is:

Table 13 - 349: Output table (columns 1–5)

hit_time_gmt	ip	visid_low	visid_high	visit_num	geo_region	geo_country	cnt	path
2012-03-01 23:04:11	72.152.194.83	2738316 554	12179707 71	1	ga	usa	1	[Cart : Billing and Shipping, Cart : Billing and Shipping,Cart]
2012-03-01 17:51:27	69.244.168.22 2	6917530 8240150 25988	28575394 33292760 267	10	mi	chn	1	[Cart : Billing and Shipping, Cart : Billing and Shipping, International]
2012-03-02 17:01:21	166.205.9.10	2059641 673	27984550 50	11	tx	usa	1	[Cart : Billing and Shipping, SEARCH , Home Page]

Store this output in the npathviz_input table.

Example SQL-MR Call 1

This example partitions the result set by geo_country and uses the ACCUMULATE clause to add the input geo_country column to the output.

```
create table NpathViz_output_table distribute by hash(geo_country) as (
  select * FROM NpathViz(
    ON npathviz_input
    PARTITION BY geo_country
    TITLE('Example 1')
    frequency_col('cnt')
    path_col('path')
    accumulate('geo_country')
  )
);
```

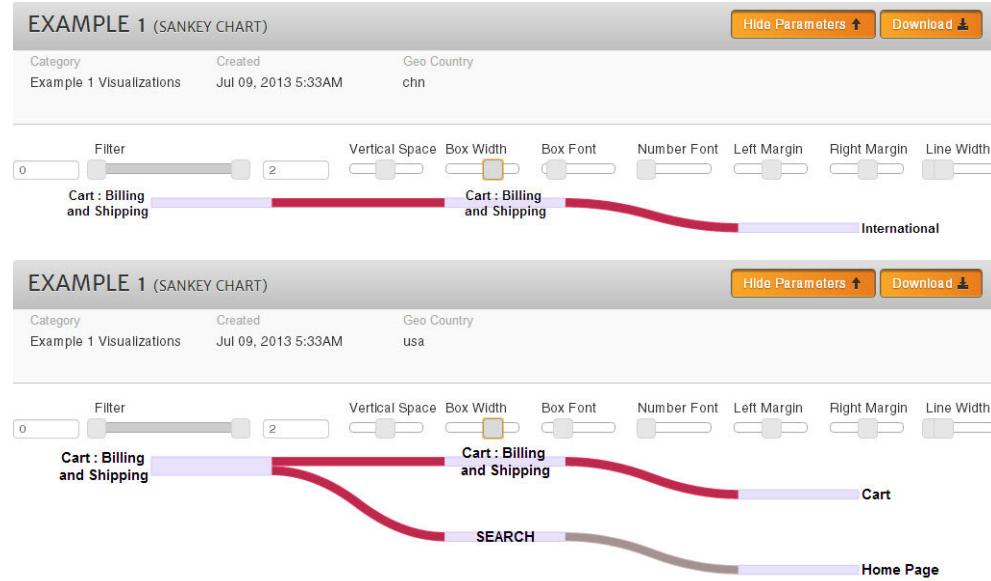
Example Output 1

```
select * from npathviz_output_table;
```

Table 13 - 350: npathviz_output_table

id	blob	time_stamp	graph_type	title	geo_country
835078848	...	2013-07-01 15:43:40.006	sankey	Example 1	usa
897606758	...	2013-07-01 15:43:40.418	sankey	Example 1	chn

Using [Aster Lens](#), the two visualizations generated by NpathViz look like this:



Example SQL-MR Call 2

This example uses two inputs and the ACCUMULATE and ARGUMENTS clauses. The second input computes aggregates on the input data and uses the same column for partitioning as the one used for the first input to avoid getting aggregates on different data.

```
create table npathviz_output_table distribute by hash(geo_country) as (
  select * FROM NpathViz(
    ON npathviz_input as input partition by geo_country
    ON (select geo_country,max(visit_num)
        from npathviz_input group by geo_country) as aggregates
    partition by geo_country
    title('Example 2')
    frequency_col('cnt')
    graph_type('sankey')
    path_col('path')
    arguments('start date=10/12/2013','end date=10/30/2013')
    accumulate('geo_country')
  )
);
```

Example Output 2

```
select * from NpathViz_output_table;
```

Table 13 - 351: NpathViz_output_table

id	blob	time_stamp	graph_type	title	geo_country	start date	end date	max(visit_num)
6537404	...	2012-03-02	sankey	Example 2	usa	10/12/2013	10/30/2013	11
1665382		08:21:23						
68112								
5127835	...	2012-03-02	sankey	Example 2	chn	10/12/2013	10/30/2013	10
3642314		08:21:47						
16269								

Example SQL-MR Call 3

This example uses the SANKEY_NODE_ORDER argument. In the Sankey diagram generated by this example, the “Home Page” node appears, vertically, before the “SEARCH” node. The order of rest of the nodes depends on the SANKEY_SORTING argument.

```
create table NpathViz_output_table distribute by hash(geo_country) as (
select * FROM NpathViz(
    ON npathviz_input
    PARTITION BY geo_country
    title('Example 3')
    frequency_col('cnt')
    path_col('path')
    sankey_node_order('Home Page', 'SEARCH')
    accumulate('geo_country')
)
);
```

Example Output 3

```
select * from NpathViz_output_table
```

Table 13 - 352: NpathViz_output_table

id	blob	timestamp	graph_type	title	geo_country
6537404166538268112	...	2012-03-02 08:21:23	sankey	Example 3	usa
6537404166538212345	...	2012-03-02 08:21:47	sankey	Example 3	chn

Example SQL-MR Call 4

This example uses the SANKEY_NODE_ORDER_ARRAY argument. In the Sankey diagram, “SEARCH” appears first and then “Cart.” Also, “International” appears first and then “Cart.” This provides a very granular approach to the ordering of nodes. Order of the rest of the nodes depends on the SANKEY_SORTING argument.

```
create table NpathViz_output_table distribute by hash(geo_country) as (
select * FROM NpathViz(
    ON npathviz_input
    PARTITION BY geo_country
    title('Example 4')
    frequency_col('cnt')
    path_col('path')
    sankey_node_order_array('2=>SEARCH::Cart', '3=>International::Cart')
    accumulate('geo_country')
)
);
```

Example Output 4

```
select * from NpathViz_output_table;
```

Table 13 - 353: NpathViz_output_table

id	blob	timestamp	graph_type	title	geo_country
6537404166538268112	...	2012-03-02 08:21:47	sankey	Example 4	usa
6537404166538212345	...	2012-02-02 08:11:00	sankey	Example 4	chn

Error Messages

- ERROR: SQL-MR function NpathViz requires argument clause: PATH_COL.
Reason: path_col not specified or is incorrectly spelled.
- ERROR: SQL-MR function NpathViz does not use argument clause: SANKEY_.
Reason: The argument clause is incorrectly spelled.
- ERROR: SQL-MR function NpathViz failed: Error, invalid justify specified: fa. Valid justify options are: [left, right].
Reason: The value of the justify argument is not correctly specified.
- ERROR: SQL-MR function NpathViz failed: Error, invalid output format specified: san. Valid formats are: [graphviz, sankey, gexf, sigma, chord, tree].
Reason: The value of the graph_type argument is not correctly specified.
- ERROR: SQL-MR function NpathViz failed: Error, order array [3=International] must have a column number and a node list, e.g. 1=>NodeA::NodeB::NodeC.
Reason: The value of the sankey_node_order_array argument is not correctly specified.
- ERROR: SQL-MR function NpathViz failed: Error, order array index [-3] must be greater than zero, e.g. 1=>NodeA::NodeB::NodeC.
Reason: A negative column number is specified in the sankey_node_order_array argument.

- ERROR: SQL-MR function NpathViz failed: Error, order array index [A] must be an integer, e.g. 1=>NodeA::NodeB::NodeC.
Reason: A character specified instead of a number in the sankey_node_order_array argument.
- ERROR: SQL-MR function NpathViz failed: Enter either 'sankey_node_order' for global ordering or enter 'sankey_node_order_array' for per stack ordering but not both.
Reason: Specified both sankey_node_order_array and sankey_node_order in the query.
- ERROR: SQL-MR function NpathViz failed: Please specify the aggregates table for NpathViz with an 'aggregates' alias and input table with an 'input' alias.
Reason: An alias for the ON clauses is not specified.

CfilterViz

Summary and Background

CfilterViz is a multiple-input partition SQL-MR function that visualizes the output of the cfilter SQL-MR function (“[Collaborative Filtering \(cfilter\)](#)” on page 418).

This function uses the [Sigma](#) visualization module to generate Sigma graphs. Additionally, this function lets you specify these types: [GEXF](#), [Graphviz](#).

Usage

Syntax (version 1.0)

```
SELECT *
FROM CfilterViz (
    ON <table|view|query> [AS 'input'] PARTITION BY i_col_name
    [ON <table|view|query> [AS 'aggregates'] PARTITION BY a_col_name]
    SCORE_COL('score_col')
    ITEM1_COL('col1_item1')
    ITEM2_COL('col1_item2')
    CNT1_COL('cnt1')
    CNT2_COL('cnt2')
    [DIRECTED('true' | 'false')]
    [GRAPH_TYPE('sigma' | 'gexf' | 'graphviz')]
    [ACCUMULATE('col_name')]
    [ARGUMENTS('key1=value1', 'key2=value2', 'key3=value3')]
    TITLE('Title')
    [SUBTITLE('Subtitle')]
)
;
```

Arguments

ON	Required	The first ON clause specifies the input table. If you specify a second input, you must use 'input' as the alias for this clause. (Optional) The second ON clause specifies a user-defined aggregate. You must use 'aggregates' as the alias for this clause.
SCORE_COL	Required	The column in the input table, outputted by cfilter, that defines the affinity strength between the nodes. Specifying 'cntb' is more appropriate than specifying 'score' as the column from cfilter's output to use (see the note below). In the input table, the 'cntb' column contains the count of the co-occurrence of two items (for example, situations where people buy two items together). CfilterViz adds those counts together. If you specify the 'score' column (contains the product of two conditional probabilities), the aggregated value calculated by CfilterViz is the sum of all of the probabilities, which results in a value greater than 1. Note: The cfilter function calculates score as the product of two conditional probabilities: $(cntb * cntb)/(cnt1 * cnt2)$, or $P(item2 item1) * P(item1 item2)$
ITEM1_COL	Required	A column with <i>item1</i> items from the cfilter output.

<i>ITEM2_COL</i>	Required	A column with <i>item2</i> items from the cfilter output.
<i>CNT1_COL</i>	Required	A column with <i>cnt1</i> counts from the cfilter output.
<i>CNT2_COL</i>	Required	A column with <i>cnt2</i> counts from the cfilter output.
<i>DIRECTED</i>	Optional	The graph mode ('true' for a directed graph; 'false' for undirected). Default is 'false'.
<i>TITLE</i>	Required	Title of the graph. Appears as a separate column in the output.
<i>SUBTITLE</i>	Optional	Subtitle of the graph. The values of the ACCUMULATE clause are appended to every partition. For example, if you partition by an ID column and accumulate that as well the ID number, this information appears in square brackets in the subtitle. Appears as a separate column in the output.
<i>GRAPH_TYPE</i>	Optional	The output graph type (default is sigma). The supported values are sigma, gexf, and graphviz.
<i>ACCUMULATE</i>	Optional	Columns from the input table that you want in your output. Note: The ACCUMULATE argument can only contain the names of the columns that have been specified in the PARTITION BY clause. If not, the output is inconsistent.
<i>ARGUMENTS</i>	Optional	The user-defined key-value pairs to be displayed in the output. The key/value pairs are case-sensitive. The function does not change their capitalization. Additionally, the function does not perform any trimming of the content. The characters to the left of the "=" operator define the column name. The characters to the right define the value inserted into the column. The value is repeated for every row generated by the function. Note: The single-quote, double-quote, and backslash characters are not allowed in keys. If you use any of these characters in a key, the function displays an error message.

Input

CfilterViz takes as input one or two tables/relations:

- Output table generated by cfilter.
- (Optional) A user-defined table containing aggregates.

Output

Depending on whether the PARTITION BY clause is used, the output of CfilterViz is a table with one or more rows. Each row contains a ZIP archive stored in the blob column. This archive contains all of the files needed to display the visualized data. In addition, the function outputs visualization metadata, including user-defined metadata.

Table 13 - 354 describes the default schema of the output table generated by CfilterViz:

Table 13 - 354: Output table schema

Column	Type
id	bigint
blob	bytea

Table 13 - 354: Output table schema (continued)

Column	Type
time_stamp	timestamp without time zone
graph_type	character varying
title	character varying

To add additional metadata columns, use the `ACCUMULATE` argument.

To view the visualizations generated by CfilterViz, use [Aster Lens](#).

Examples

Example Input

The examples in this section use as input the CfilterViz_data table, which contains example cfilter output:

```
select * from cfiltersviz_data limit 5;
```

Table 13 - 355: CfilterViz_data

storeid	col1_item1	col1_item2	cnty	cnt1	cnt2	score	support	confidence	lift	z_score
2	butter	eggs	1	1	3	0.333 33333 33333 33	0.3333 333333 333333 33	1	1	-0.577350269189626
2	butter	flour	1	1	2	0.5 333333 33333	0.3333 333333 33333	1	1.5	-0.577350269189626
2	eggs	butter	1	3	1	0.333 33333 33333 33	0.3333 333333 333333 33	0.33333333 3333333	1	-0.577350269189626
2	eggs	milk	1	3	1	0.333 33333 33333 33	0.3333 333333 333333 33	0.33333333 3333333	1	-0.577350269189626
2	eggs	flour	2	3	2	0.666 66666 66666 67	0.6666 666666 666666 66667 67	0.66666666 6666667	1	1.73205080756888

Example SQL-MR Call 1

This example partitions by 1.

```
create table CfilterViz_output_table1
distribute by hash(storeid) as (
    select * from CfilterViz(
        on cfilterviz_data
        partition by 1
        title('CfilterViz Example 1')
        score_col('cntb')
        item1_col('col1_item1')
        item2_col('col1_item2')
        cnt1_col('cnt1')
        cnt2_col('cnt2')
        accumulate('storeid')
    )
)
;
```

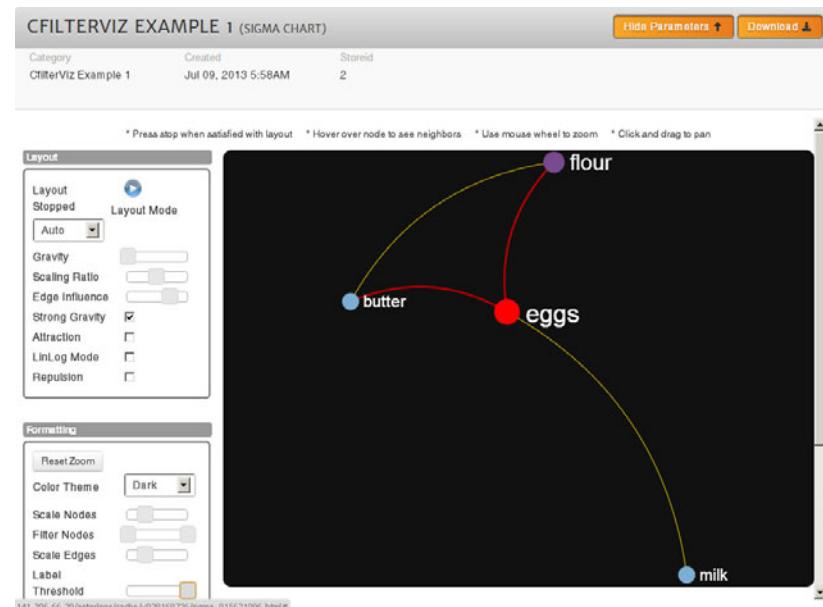
Example Output 1

Table 13 - 356: Output table

id	blob	time_stamp	graph_type	title	storeid
653269008	...	2013-04-19 10:08:06.087	sigma	CfilterViz Example 1	2

The visualization data is in the “blob” column.

Using [Aster Lens](#), the visualization looks like this:



Example SQL-MR Call 2

This example partitions the results by storeid.

```
create table CffilterViz_output_table2 distribute by hash(storeid) as (
select * from CffilterViz(
    on CffilterViz_data
    partition by storeid
    title('CffilterViz Example 2')
    subtitle('Distributed and ordered by storeid')
    score_col('cntb')
    item1_col('col1_item1')
    item2_col('col1_item2')
    cnt1_col('cnt1')
    cnt2_col('cnt2')
    accumulate('storeid')
))order by storeid;
```

Example Output 2

Table 13 - 357: Output table

id	blob	time_stamp	graph_type	title	subtitle	Storeid
5972686530	...	2013-04-19 10:36:11.963	sigma	CfilterViz Example 2	Distributed and ordered by storeid	1
3466227494	...	2013-04-19 10:36:13.825	sigma	CfilterViz Example 2	Distributed and ordered by storeid	2
1589605940	...	2013-04-19 10:35:24.474	sigma	CfilterViz Example 2	Distributed and ordered by storeid	3

Example SQL-MR Call 3

This is an example of a multi-input query:

```
create table aster_lens.CfilterViz_output_table3 distribute by
hash(storeid) as (
select * from CffilterViz(
    on aster_lens.CfilterViz_data as input
    partition by storeid
    on (select storeid,max(support) as max_sup
    from aster_lens.CfilterViz_data group by storeid) as aggregates
    partition by storeid
    title('CfilterViz Example 3')
    subtitle('Distributed and ordered by storeid--multiple input')
    score_col('cntb')
    graph_type('sigma')
    item1_col('col1_item1')
    item2_col('col1_item2')
    cnt1_col('cnt1')
    cnt2_col('cnt2')
    arguments('start_date=10/12/2013','end_date=10/30/2013')
    accumulate('storeid'))
)order by storeid;
```

Example Output 3

Table 13 - 358: Output table

id	blob	time_stamp	graph_type	title	subtitle	storeid	end_date	start_date	max_sup
2525	...	2013-04-19	sigma	CfilterViz	Distributed	1	10/30/	10/12/	0.75
3005		10:48:10.125		Example 3	and ordered by storeid-- multiple input		2013	2013	
65									
1744	...	2013-04-19	sigma	CfilterViz	Distributed	2	10/30/	10/12/	0.666666
3240		10:48:57.658		Example 3	and ordered by storeid-- multiple input		2013	2013	6666666
36									67
8106	...	2013-04-19	sigma	CfilterViz	Distributed	3	10/30/	10/12/	1
4608		10:49:00.447		Example 3	and ordered by storeid-- multiple input		2013	2013	

Visualization Function Usage Notes

Directed versus Undirected Graphs

Using the *DIRECTED* argument, you can specify whether graphs are directed or undirected. To understand how NpathViz and CfilterViz handle directed and undirected graphs, consider this cfilter example output:

Table 13 - 359: cfilter example output

item_1	item_2	score
A	B	0.5
B	A	0.2

If you specify DIRECTED('true'), CfilterViz produces two edges:
A->B score 0.5 and B->A score 0.2

If you specify DIRECTED('false'), CfilterViz produces one edge:
A-B score 0.7.

Thus, in an undirected graph, A->B and B->A mean the same thing and their scores are added together. Visualizations of directed graphs tend to be busier in appearance because they have either one or two edges connecting any two nodes. Undirected graphs always have one edge connecting two nodes. We recommended that you only use directed graphs when direction is very important or one direction significantly dominates the reverse direction.

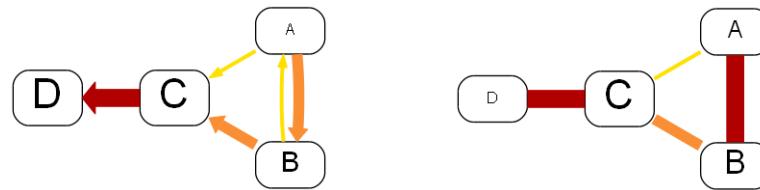
Graph Representation of nPath

You can represent nPath accumulated strings on a graph, either directed or undirected. To understand how NpathViz does this, consider this nPath example:

Table 13 - 360: nPath example

path	count
[A, B, C, D]	10
[B, A, C, D]	5

NpathViz breaks the paths into pairs (for example, A, B, C, D becomes A, B ; B, C ; C, D. The rules for the directed/undirected settings are as described in [Directed versus Undirected Graphs](#). The resulting graphs, directed and undirected, look like this:



Chord Diagrams

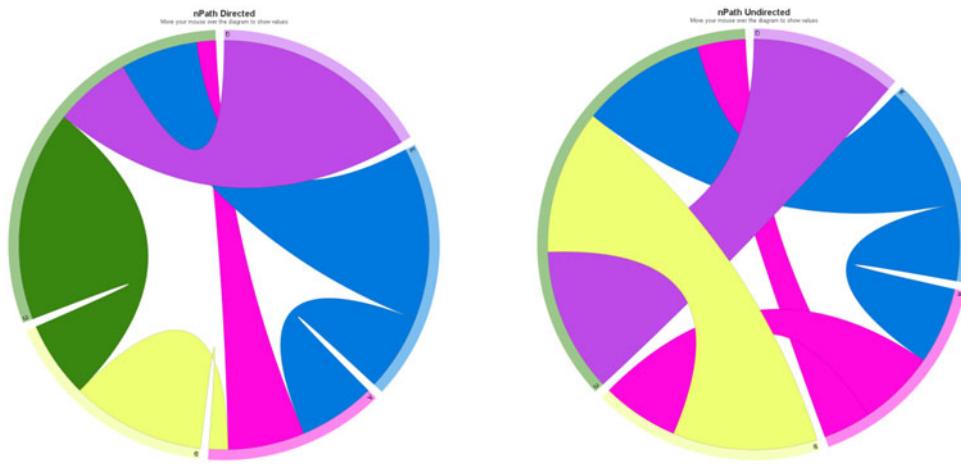
Another way to visualize a graph is using a chord diagram. This is best when the number of nodes is small, say under 50, and you want to show direction (DIRECTED='true')).

The edges are calculated in the same as the example in [Graph Representation of nPath](#). The nodes are sized by the chord visualization engine to match the sum of the chord sizes. In this sense, the nodes in the chord and graph visualizations are sized in a similar fashion.

Below are example chord diagrams for directed and undirected graphs. Notice that in the directed diagram, chords can be asymmetric whereas in the undirected diagram, chords are always the same size on both ends.

Table 13 - 361: nPath example output

path	count
[A, B, C, D]	10
[C, B, C, A, E, C, D, C, E, A]	20
[C, D]	20
[B, A, C, D]	5
[A, B, C, E]	25



Sankey Representation of nPath

NpathViz can also represent nPath accumulated paths using a Sankey flow diagram. Consider this example, similar to the one above, but with additional data:

Table 13 - 362: Example nPath output

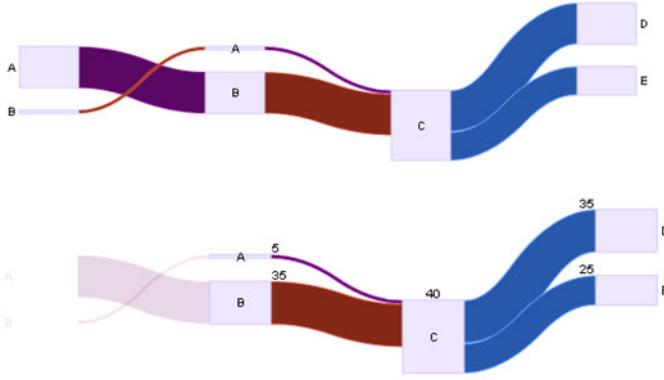
path	count
[A, B, C, D]	10
[B, A, C, D]	5
[C, D]	20
[A, B, C, E]	25

Table 13 - 362 shows the effects of using the JUSTIFY argument on four unique paths that were followed. The count indicates how many times that exact path was followed.

Table 13 - 363: Sankey justification examples

Justification	Example
JUSTIFY('left') Note: The two diagrams are the same, but the second shows the line weights for a segment of the graph. The line weights are displayed when you hover the mouse over that segment.	

Table 13 - 363: Sankey justification examples (continued)

Justification	Example
JUSTIFY ('right') Note: The two diagrams are the same, but the second shows the line weights for a segment of the graph. The line weights are displayed when you hover the mouse over that segment.	

Note the following about the examples in [Table 13 - 363](#):

- Path C, D is shorter than the rest. You can choose if this path should be left- or right-justified, as shown in the above diagrams. Notice how in the right-justified diagram, “C, D” with a count of 20 is at position 3 and 4, and added to the existing C, D in the same spot (from A,B,C,D and B,A,C,D).
- The first and last rows are “browsing cousins,” meaning that although 10 people followed one and 25 people followed the other, they are very similar. It could be said 35 people followed A,B,C and branched off at the end. Sankey diagrams capture this importance nuance.
- Similarly, the unique paths are intentionally lost. In other words, you do not see A,B,C,D and A,B,C,E as separate paths. Instead, they are merged when they overlap.
- The DIRECTED argument is ignored in Sankey diagrams. Direction always flows from left to right.

Sankey Edge (Line) Color

Sankey edge (line) color is determined as follows:

A given node name (for example, “Home Page”) always has the same color. Thus, if you generate a Sankey diagram containing the Home Page node today or next week, the node gets the same color. The color is based on the hash value of the node label. This consistency makes it possible to compare multiple Sankey diagrams since the human eye gravitates towards like colors.

For a given node, the color is the same for all edges to the right of the node.

If a node appears in multiple Sankey stacks (columns), it always gets the same color edges. This makes it possible to quickly spot repeating paths.

Sankey Node Order

There are various reasons why you might want to control the node order in Sankey diagrams:

- Comparing Sankey diagrams: If you are comparing, say, abandoned vs non-abandoned shopping carts, you want to look at discrepancies in the Sankey paths on the two diagrams. This is only possible if the two diagrams look almost identical. That is, their nodes must be in the same order on both diagrams.
- Looking for a preferred path: If you want to look at a certain path, say a checkout sequence, and see how people deviate from it. You can put the preferred path as the first node in each sankey stack. For example, consider this path: Begin Checkout -> Enter address -> enter credit card -> submit; these are all nodes at the very top of each Sankey stack.
- Formatting cleanliness: If you have, say, a conversion event at the far right side of the Sankey diagram, and it's the only node in that sankey stack, you want it to be relatively centered. Being able to control the node order allows that as opposed to auto placement.

To specify node order, use these arguments:

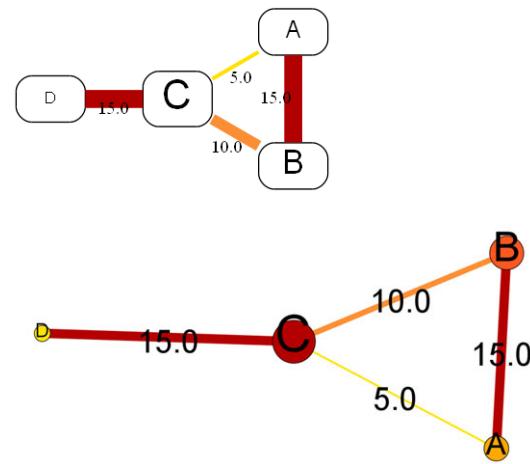
- SANKEY_SORTING: A value of 'true' means you want the Sankey nodes sorted alphabetically. If you are comparing two Sankey diagrams, you probably want to set this to 'true'.
- SANKEY_NODE_ORDER('D','B','C','A'): specifies a comma-separated, quoted list of sankey node names. The sankey nodes (for all columns) appears in the specified order. Nodes not specified get sorted according to the value of the SANKEY_SORTING argument.
- SANKEY_NODE_ORDER_ARRAY('1=>D::B','3=>C::B','2=>F:D::B'): This argument is for the power user who wants very granular control over node order for all nodes. You can specify an array with a column number => the node list, as shown in the example. This lets you have a different node order per Sankey column. Whatever you omit gets sorted according to the value of the SANKEY_SORTING argument. Column 1 is the left most column.

Node and Edge Presentation

Node size/color and edge size/color is computed based on an accumulated score. Consider [Table 13 - 360](#).

Node C is the largest, as shown in the two graphs below (Graphviz and GEXF). C gets credit for edge scores $15+5+10=30$. B gets credit for edge scores $10+15=25$, and so on. In the case of directed graphs, nodes gets credit when they are the target.

Edge thickness is computed based on whether the graph is directed/undirected, as explained above. Edge thickness is larger and darker for higher-edge scores. Node size is larger and darker for higher-node scores.



Tree Representation of nPath

Similar to Sankey, nPath can be represented as a tree by specifying GRAPH_TYPE('tree'). Trees can be left- or right-justified like Sankey. The tree is different than Sankey because trees show all discrete paths followed whereas Sankey diagrams aggregate common path segments.

When the tree is first opened in the browser, the most dominant path is opened by default. The end user can open/close any path interactively.

CHAPTER 14 Aster Database System Utility Functions

The built-in system utility functions are intended to be invoked through AMC Executables. These functions are automatically installed as part of the Aster Database installation. Note that if you type \df in ACT, these out-of-the-box functions will not appear, as they are internal-only functions. You can, however, use these in your own custom scripts.

Aster Database includes the following system utility functions.

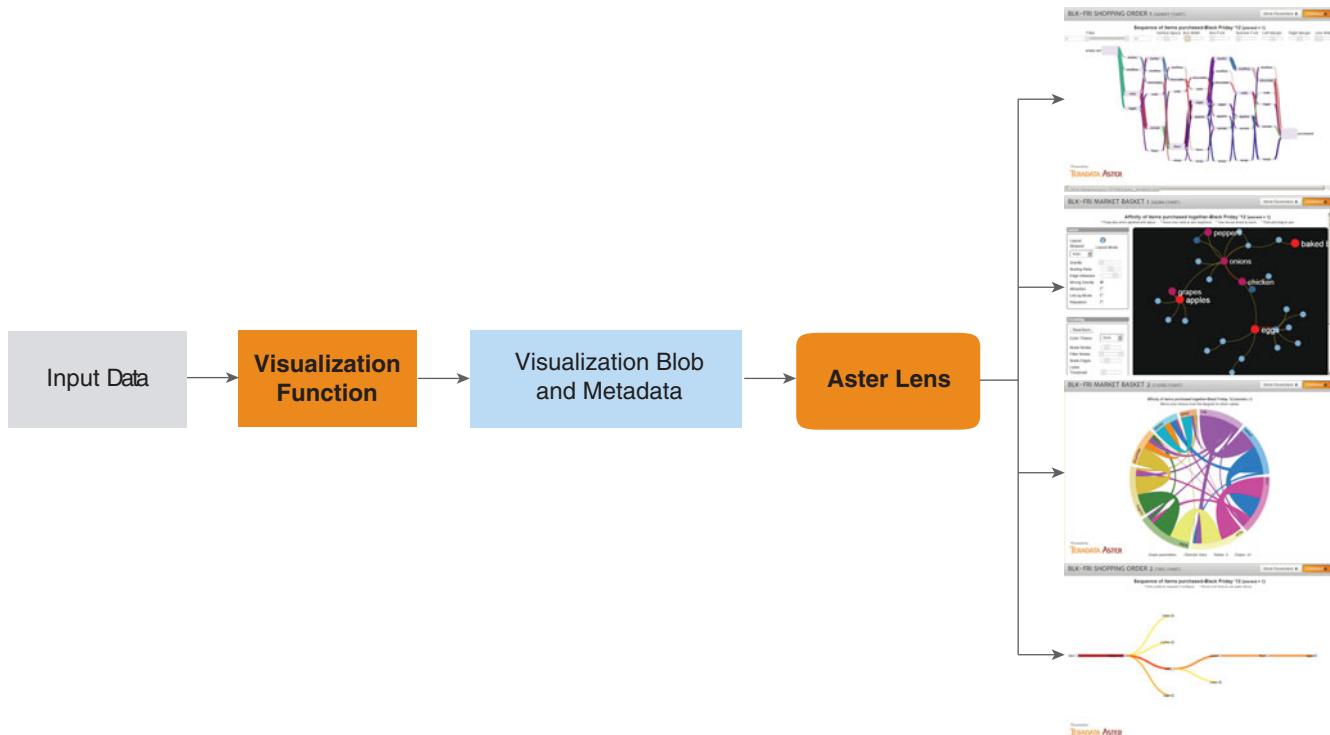
- nc_skew
- nc_relationstats

For more information on these functions, see one of these documents, depending on your platform:

- For appliances, see the *Teradata Aster Big Analytics Appliance 3H Database User Guide*.
- For installations on commodity hardware, see the *Aster Database User Guide*.

CHAPTER 15 Aster Lens

Aster Lens is a web-based application for viewing visualizations created by Teradata Aster's SQL-MR visualization functions.



- [Aster Lens Concepts](#)
- [SQL-MR Visualization Functions](#)
- [Installing and Setting Up Aster Lens](#)
- [Using Aster Lens](#)

Aster Lens Concepts

Aster Lens relies on Visualization Tables and Visualization Catalog Tables to reference the visualizations stored in Aster Database.

- [How Aster Lens Works](#)
- [Visualization Tables](#)
- [Visualization Catalog Tables](#)
- [Aster Lens Architecture](#)
- [User Authentication](#)

How Aster Lens Works

To view visualizations in Aster Lens, you must store the output of SQL-MR visualization functions in [Visualization Tables](#). You must also make sure that those tables are referenced in the [Visualization Catalog Tables](#) that Aster Lens is configured to use.

When you log in to Aster Lens from a client system, Aster Lens uses the Visualization Catalog Tables in the Aster Databases it is configured to access to find the available visualizations.

Aster Lens lists those visualizations in its GUI. When you choose to view a visualization, Aster Lens renders it on your system's Web browser.

Visualization Tables

To view visualizations in Aster Lens, you must store the output of SQL-MR visualization functions in Visualization Tables. Each row in a Visualization Table contains a blob column where the visualization is stored and columns containing visualization metadata information.



One call to a visualization function can generate multiple rows of output (one row for each partition).

Visualization Tables let you organize your visualizations and provide Aster Lens with the information it needs to display them. You can create multiple Visualization Tables, each representing a visualization "category." Visualization Tables can have different metadata columns.

Visualization Table Schema

[Table 15 - 364](#) describes the default schema of a Visualization Table.

Table 15 - 364: Visualization Table schema

Column	Type	Description
id	bigint	Unique identifier.
blob	bytea	A blob of data representing a ZIP archive containing the visualization.
time_stamp	timestamp without time zone	The time when the visualization was created.
graph_type	character varying	The visualization type (Chord, Sankey, Tree, Sigma, GEXF, and Graphviz)
title	character varying	The visualization title.

When you create a Visualization Table, you can use the default schema or add additional columns to it using the ACCUMULATE and ARGUMENTS clauses, as shown in the examples below.

Example: Adding Visualizations To a Visualization Table

One way to create a Visualization Table is to use the CREATE TABLE statement when creating the first visualization. Then, you can use the INSERT INTO TABLE statement to add visualizations to the table.

Creating a Visualization Table

This example creates the Visualization Table aster_lens.cart_abandonment and adds rows to it containing the visualization of the data outputted by nPath and stored in the aster_lens.npath_output_abandoned_shopping_order table.

```
create table aster_lens.cart_abandonment distribute by hash(id) as
  (select * FROM nPathViz(
    ON aster_lens.npath_output_abandoned_shopping_order as input
    partition by storeid
    frequency_col('cnt')
    graph_type('sankey')
    path_col('path')
    arguments('start_date=10/12/2013','end_date=10/30/2013',
              'owner=ASTER','tags=Coupon Sale')
    title('Shopping Order')
    subtitle('Tracking order in which items added to cart')
    accumulate('storeid'))
) ;
```

In this example, the ARGUMENTS clause specifies user-defined metadata (start_date, end_date, owner, and tags). This information appears on the back of the visualization card in **Card View**.



This information also appears in **Table View**. In addition, the column names specified by the ACCUMULATE clause appear in **Table View** along with their corresponding values.

Abandoned Carts Example						
	Title	Subtitle	Category	Created	Start Date	
	Shopping Order	Tracking order in which items added to cart	Abandoned Carts Example	Jul 01, 2013 12:06PM	10/12/2013	
	End Date	Tags	Storeid	Owner		
	10/30/2013	Coupon Sale	1	ASTER		

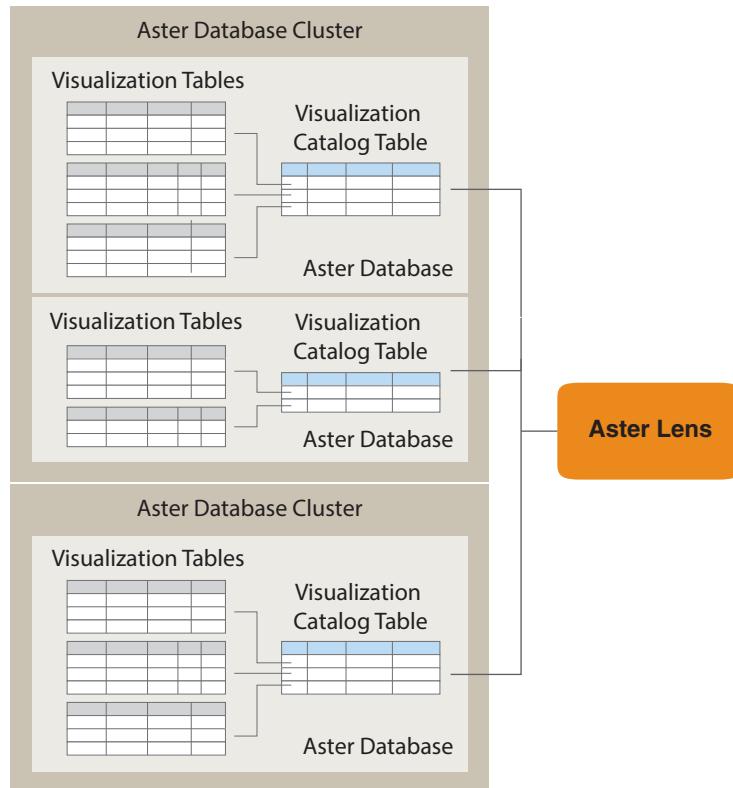
Adding Rows To a Visualization Table

This example adds rows to the Visualization Table aster_lens.cart_abandonment. The rows contain the visualization of the data outputted by a second running of nPath on a different set of data.

```
insert into aster_lens.cart_abandonment
  select * FROM nPathViz(
    ON aster_lens.npath_output_abandoned_shopping_order as input
    partition by storeid
    frequency_col('cnt')
    graph_type('tree')
    path_col('path')
    arguments('start_date=10/12/2013','end_date=10/30/2013',
              'owner=ASTER','tags=Coupon Sale')
    title('Shopping Order')
    subtitle('Tracking order in which items added to cart')
    accumulate('storeid')
  ) ;
```

Visualization Catalog Tables

A Visualization Catalog Table is used by Aster Lens to reference the Visualization Tables in an Aster Database. Each row in a Visualization Catalog Table points to a Visualization Table. Aster Lens uses one Visualization Catalog Table per database, as shown in this example.



Here is an example of a Visualization Catalog Table:

Table 15 - 365: Visualization Catalog Table example

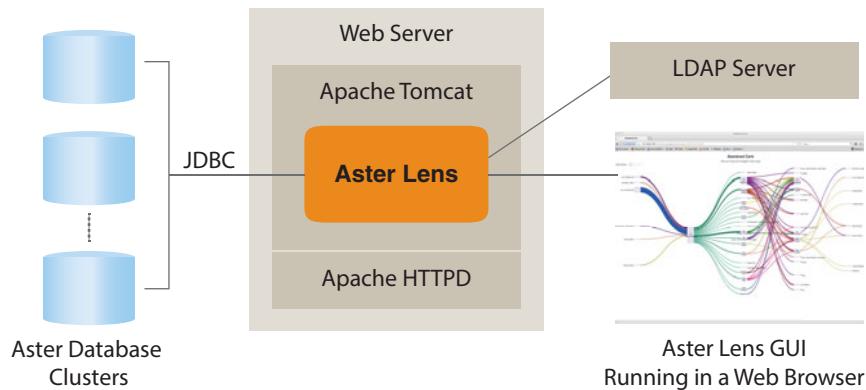
id	schema_name	table_name	category_name
1	aster_lens	cart_abandonment	Abandoned Carts Example
2	aster_lens	sales_completed	Completed Sales Example

By default, the name of a Visualization Catalog Table is `aster_lens_catalog` and is stored in the public schema of an Aster Database. However, you can use a different name as described in [Configuring Aster Cluster Settings](#).

Aster Lens Architecture

Aster Lens is a web application that runs in a web server. The components required for running Aster Lens are:

- Apache Tomcat—A web application server platform on which Aster Lens runs. The version of Apache Tomcat included with Aster Lens is 7.0.35.
- Apache Hypertext Transfer Protocol Server (HTTPD)—An HTTP web server that Aster Lens uses to receive and process visualization requests. The version of HTTPD included with Aster Lens is 2.2.23.
- Lightweight Directory Access Protocol (LDAP) Server—(Optional) A server that provides authentication services. If an LDAP server is not available, only the built-in “admin” user can access Aster Lens. Aster Lens supports any version of LDAP that is properly configured. For more information, [Configuring Aster Lens LDAP Settings](#).
- Aster Database Clusters—The clusters where the visualizations are stored.
- JDBC—The Java-based data access protocol used by Aster Lens to access Aster Database.
- Web browser—A supported web browser that runs the Aster Lens GUI.



User Authentication

Aster Lens provides two authentication options:

- LDAP authentication—This option allows you to use your existing LDAP infrastructure for user authentication.
- Built-in “admin” user—The built-in user admin has full access to Aster Lens.

SQL-MR Visualization Functions

This version of Aster Analytics Foundation provides these SQL-MR visualization functions:

- [NpathViz](#) —Visualizes the output of Teradata Aster nPath.
- [CfilterViz](#) —Visualizes the output of cfilter.

Installing and Setting Up Aster Lens

- [System Requirements](#)
- [Installing Aster Lens](#)
- [Configuring Aster Lens Properties](#)
- [Working with Visualization Catalog Tables](#)
- [Installing and Uninstalling Examples](#)
- [Starting and Stopping Aster Lens](#)
- [Uninstalling Aster Lens](#)
- [Manually Installing Aster Lens](#)
- [Installing Aster Lens on the Queen](#)

System Requirements

It is recommended to install Aster Lens on a single, dedicated Linux node that is not part of the Aster Database cluster.

System requirements:

- At least 4 GB of memory
- At least 50 GB of disk space
- A 64-bit CPU if using the bundled Apache server
- Operating systems:
 - Redhat 6.0 and 6.2
 - SLES 11 SP2
- At least one Aster Database cluster
- (Optional) LDAP server

Supported browsers:

- Firefox v19.0 and higher
- Safari v5.1.7 and higher
- Google Chrome v27.0 and higher (full support for Aster Lens, but no support for viewing downloaded visualizations)
- Internet Explorer v10 (supports Aster Lens, but will not display Sigma graphs. Chord graphs are supported, but they are not interactive).

When attempting to open locally-stored HTML files that use AJAX technology, as in the case of downloaded visualizations, some browsers do not display the content because they do not allow access to local system files for security reasons:

- Internet Explorer 10 issues a warning.
- Google Chrome does not display the content at all.
- Safari and Firefox allow the content to be displayed normally.

Installing Aster Lens



The installation requires root access. Use the sudo command when needed.

To install Aster Lens on a clean machine (operating system installed, but no web server), use the Teradata Aster provided package `AsterLensInstaller_5-10.bin`. This is the recommended configuration. If you wish to install on a machine that already has a web server or on the Aster queen node, use one of these procedures instead:

- For a machine with Apache web server already installed: “[Manually Installing Aster Lens](#)” on page 564
- For installing on the Aster queen (not recommended except for testing or demo purposes): “[Installing Aster Lens on the Queen](#)” on page 566

This `AsterLensInstaller_5.10.bin` installer package:

- Installs the Aster Lens files in the `/opt/AsterLens` directory.
- Creates the “toolchain” symbolic link in the `/home/beehive` directory. This steps is required for launching the bundled Apache Web Server.
- For every Aster Database specified in the `/opt/AsterLens/asterlens/asterlens.properties` file, installs the specified Visualization Catalog Table.
- (Optional) Installs the examples included with Aster Lens (you can remove them later by running the `uninstall-examples.sh` script). If you choose not to install the examples, you can install them later, as described in [Installing Examples](#).
- (Optional) Starts Aster Lens. If you choose not to start Aster Lens, you can always start it using the `start-asterlens.sh` script.

To install Aster Lens:

1 Copy `AsterLensInstaller_5-10.bin` to the system on which you want to install Aster Lens.

2 Make `AsterLensInstaller_5-10.bin` executable.

```
# sudo chmod +x AsterLensInstaller_5-10.bin
```

3 Run `AsterLensInstaller_5-10.bin` as root:

```
# sudo ./AsterLensInstaller_5-10.bin
```

This command installs the Aster Lens files in the `/opt/AsterLens` directory and starts the installation script `install-asterlens.sh`.



The installer adds a third-party license file (`license.html`) to the `/opt/AsterLens` directory.

4 Follow the installation instructions to install Aster Lens:

a When prompted to configure the settings in the `asterlens.properties` file, open a new command line window and edit the file as described in [Configuring Aster Lens](#)

[Properties](#). Then, return to the window where `AsterLensInstaller_5-10.bin` is running.

- b** When prompted to proceed with the installation, enter `y`.

Are you ready to proceed with the installation (y/n) :

- c** When prompted whether to set up the Apache Web Server included with Aster Lens, enter `y` to proceed with automatic setup.

Would you like to continue with this automated setup? (y/n)

After setting up the Apache Web Server, the installation script creates the “asterlens” user and “asterlens” group if they do not exist. Then, the installation script lists the installation tasks and prompts you whether to continue.

To manually set up the Apache Web Server, enter `n`. To manually install Aster Lens and set up the Apache Web Server, see [Manually Installing Aster Lens](#).

- d** When prompted whether to continue, enter `y`.

Do you want to continue? (y/n)

- e** When prompted whether to install the Aster Lens examples, enter `y` to install them. If you enter `n`, you can always install them later as described in [Installing Examples](#).

Example installation may take a few minutes.

- f** When prompted whether to start Aster Lens, enter `y`. If you enter `n`, you can always start Aster Lens later using the `AsterLens/start-asterlens.sh` script.

Starting Apache ...

Starting Tomcat ...

Aster Lens is now running on 'example-SUSE-AsterLens'. Point your browser to this host to view Aster Lens

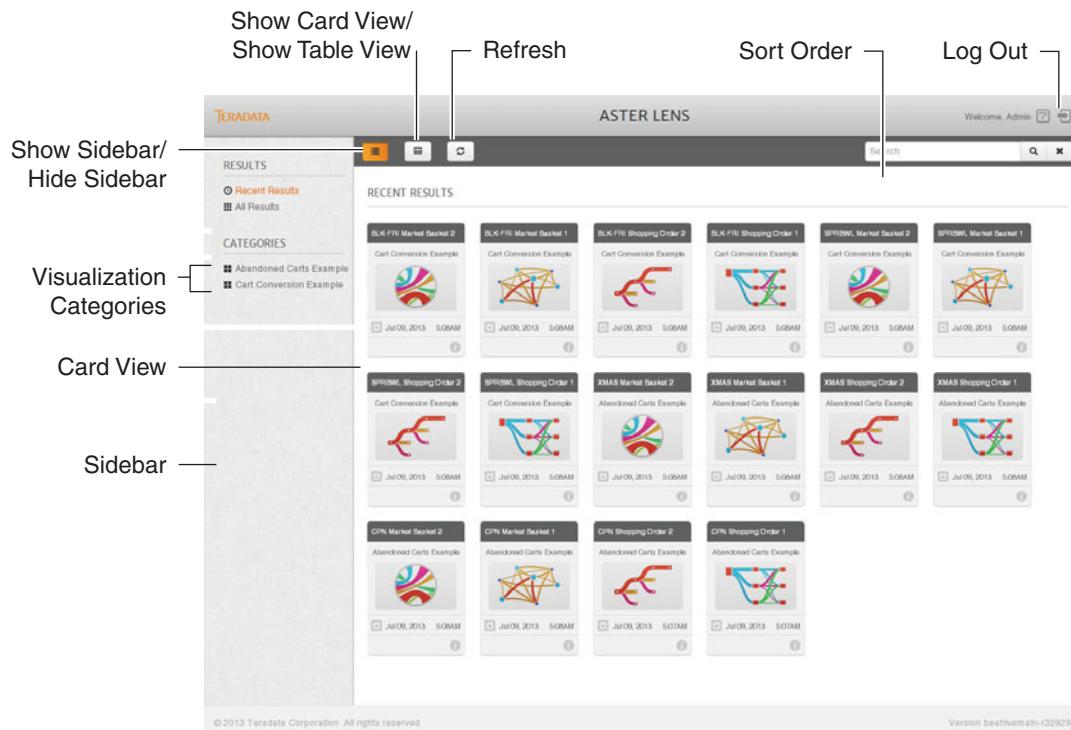
Installation complete.

After Aster Lens is configured and installed, the installation script creates symbolic links in the appropriate folders in `/etc/rc.d`. This allows Aster Lens to gracefully handle server shutdowns and failures.

- 5** To access the Aster Lens web-based GUI, open a supported browser and, in the URL field, enter the hostname of the system on which Aster Lens is running.

- 6** To log in to Aster Lens, enter the password for the admin user, which is specified in the `asterlens.properties` file. By default, the password is `admin`.

After you log in, if you chose to install the examples during the installation process, Aster Lens displays the example visualizations in **Card View**. In **Card View**, each card represents a visualization.



- 7** Click a card to show its corresponding visualization.

For more information about using Aster Lens, see [Using Aster Lens](#).

Configuring Aster Lens Properties

The `/opt/AsterLens/asterlens/asterlens.properties` file specifies the configuration settings of Aster Lens. This file consists of these sections:

- Built-in Admin Password

Although it is optional, we recommend that you change the password for the admin user.

- LDAP Settings

By default, the LDAP settings are disabled. If you do not configure the LDAP settings, the only user allowed to access Aster Lens is the “admin” user.

- Aster Cluster Settings

To use Aster Lens, you must provide the required information for at least one Aster Database on which the `NpathViz` and `CfilterViz` functions are installed.



You should restrict the ownership and access rights to the configuration file only to administrators.



After completing the Aster Lens installation, if you modify the asterlens.properties file, restart Aster Lens so that the changes take effect.

- [Configuring Aster Lens Administrator Password](#)
- [Configuring Aster Lens LDAP Settings](#)
- [Configuring Aster Cluster Settings](#)

Configuring Aster Lens Administrator Password

The Built-in Admin Password defines the password for the admin user. By default, the password is “admin”. In the asterlens.properties file, the listed password is the encrypted version of the password.

```
#####
#           Built-in Admin Password          #
#####

##### default admin user name is "admin"
adminPwd=0c754d7d6eb9db082c2b725f26ee958461a451b8a310052c66e698d7b7553805
```

To change the password:

- 1 Use the AsterLens/scripts/encrypt-password.sh script to encrypt the new password, as described in [Encrypting Passwords](#).
- 2 In the asterlens.properties file, change the value of adminPwd to the encrypted password you generated in the previous step.

Because the encrypted password is very long, use copy and paste to assign the new value of adminPwd.

Configuring Aster Lens LDAP Settings

The LDAP settings section of asterlens.properties defines the parameters that allow Aster Lens to access the LDAP server in your organization. By default, this section is commented out, which means that the settings are ignored.

```
#####
#           LDAP Settings          #
#####

##### Uncomment LDAP settings and update in order to use LDAP:
#ldapServerURL=ldap://168.65.52.91:389/dc=td,dc=teradata,dc=com
#ldapAdminUserDn=cn=admin,dc=td,dc=teradata,dc=com
#ldapAdminPwd=password

#####   default values. Un-comment to specify custom values   #####
# ldapUserDnPatterns=uid={0},ou=People
# ldapGroupRoleAttribute=cn
# ldapRolePrefix=ROLE_
# ldapGroups=ou=Groups
```

To configure Aster Lens to use the authentication services of an LDAP server, uncomment the LDAP parameters and specify their values:

Table 15 - 366: LDAP parameters

Parameter	Description
ldapServerURL	URL of the LDAP server.
ldapAdminUserDn	User name of the LDAP server administrator.
ldapAdminPwd	Password of the LDAP server administrator user account.
ldapUserDnPatterns	The pattern used to construct the user DN.
ldapGroupRoleAttribute	The information about the role name for a group.
ldapRolePrefix	The role name prefix.
ldapGroups	The LDAP groups the user has access to.

LDAP Server Installation Tips

1 Install LDAP.

2 Set up LDAP as follows:

- LDAP URL: `ldap://<ip address>:389/dc=maxcrc, dc=com`
- In this domain (`dc=maxcrc, dc=com`), you can set up "ldapAdminUserDn" as `dc=Manager, dc=maxcrc, dc=com`. This is your rootDN.
- Set the root-distinguished password.
- Create an organizational unit (`ou=People`) in the domain.
- Add a user to this OU and set its password.
- Create an organizational unit `ou=Groups`.
- Add a group “user” to the above created `ou=Groups`.
- Now add the user to the user group.

Configuring Aster Cluster Settings

The Aster Cluster Settings section in `asterlens.properties` defines the settings that Aster Lens uses to connect and access your Aster Databases. In addition, this section lets you specify which columns to exclude from Visualization Tables.

```
#####
#          Aster Cluster Settings      #
#####

##### set profile values for the desired database
aster.profileNames=Default_DB_Profile
aster.host1=10.80.163.130
aster.db1=beehive
aster.user1=db_superuser
aster.password1=655a211509cbb4d3b4d40d1aa38b1faf84e8b732b5fa7f9a9ae0c06bef0e3abb
aster.catalogs1=aster_lens_catalog
```

```
##### For additional Aster DB connection add a suffix number to all property names #####
#aster.profileNames=Secondary_DB_profile
#aster.host2=222.222.222.222
#aster.db2=beehive
#aster.user2=db_superuser
#aster.password2=655a211509cbb4d3b4d40d1aa38b1faf84e8b732b5fa7f9a9ae0c06bef0e3abb
#aster.catalogs2=aster_lens_catalog

#comma separated list of column names to exclude from visualization reports
#aster.visualization.exclude.columns=author
```

Configuring Aster Lens to Access Aster Databases

The Aster Cluster Settings section in `asterlens.properties` provides a profile for every Aster Database that Aster Lens can access. [Table 15 - 367](#) describes the profile parameters, where *n* is the profile identifier.

Table 15 - 367: Aster cluster settings

Parameter	Description
<code>aster.profileNames<i>n</i></code>	(Optional) A descriptive name of the profile.
<code>aster.host<i>n</i></code>	The IP address of the host on which Aster Database runs.
<code>aster.db<i>n</i></code>	The database name.
<code>aster.user<i>n</i></code>	The user name.
<code>aster.password<i>n</i></code>	The encrypted password. To encrypt the password, use the <code>encrypt-password.sh</code> script, as described in Encrypting Passwords .
<code>aster.catalogs<i>n</i></code>	(Optional) The name of the Visualization Catalog Table, which holds references to the Visualization Tables containing the visualizations. The default name is <code>aster_lens_catalog</code> , but you can change it.

The profile identifier for the first profile must always be 1. The identifier for the second profile must be 2, for the third profile it must be 3, and so on. Every time you add a profile, increment the profile identifier by 1. If you remove a profile, readjust the profile identifiers so that they start at 1 and are incremented by 1.

For example, this section defines the settings for three Aster Databases on three different Aster Database clusters:

```
#####
#          Aster Cluster Settings          #
#####
# Web Traffic-Tracking Database
aster.host1=168.111.111.111
aster.db1=beehive
aster.user1=db_superuser
aster.password1=655a211509cbb4d3b4d40d1aa38b1faf84e8b732b5fa7f9a9ac0c06bef0e3abb
aster.catalogs1=aster_lens_catalog

# Customer Complaint Database
aster.host2=168.222.222.222
aster.db2=beehive
aster.user2=db_superuser
```

```
aster.password2=655a211509ccb4d3b4d40d1aa38b1faf84e8b732b5fa7f9a9ae0c06bef0b3abb
aster.catalogs2=aiv_catalog

# Sales Database
aster.host3=168.333.333.333
aster.db3=beehive
aster.user3=db_superuser
aster.password3=655a211509ccb4d3b4d40d1aa38b1faf84e8b732b5fa7f9a9ae0c06bef0e3abb
aster.catalogs3=sales_DB_catalog
```

Specifying Columns to Exclude from Visualization Tables

To specify the Visualization Table columns that you want Aster Lens not to display in its GUI, set the value of the aster.visualization.exclude.columns parameter in asterlens.properties to a comma separated list of the names of those columns. This applies to all databases specified in the Aster Cluster Settings section.

For example, this line excludes the author and location columns:

```
aster.visualization.exclude.columns=author,location
```

Encrypting Passwords

The AsterLens/scripts/encrypt-password.sh script lets you encrypt passwords so that you can use them in the asterlens.properties file.

To encrypt a password:

- 1 Use the encrypt-password.sh script to encrypt the password.

For example, to encrypt the string “admin123”, enter this command:

```
# ./encrypt-password.sh admin123
```

The output of this command looks like this:

```
8eaf63397062c9624b968faab8b1f0e19c3d8640cd3ada8db6967b26ad775af6
```

- 2 Copy the encrypted password and paste it where appropriate in the asterlens.properties file.

Working with Visualization Catalog Tables

- [Creating Visualization Catalog Tables](#)
- [Adding Entries to Visualization Catalog Tables](#)
- [Removing Visualization Catalog Tables](#)

Creating Visualization Catalog Tables

When you run the Aster Lens installation script, it creates a Visualization Catalog Table in every Aster Database specified in the asterlens.properties file (one Visualization Catalog Table per database).

Later, if you add more Aster Databases to the asterlens.properties file, run the AsterLens/scripts/install-catalog.sh script to create Visualization Catalog Tables in the new databases.

For example:

```
# sudo ./install-catalog.sh

##### Step 2: Install Aster Lens catalog table(s) in aster
database(s) #####
Creating catalog table(s) for all database(s) ...

Creating catalog table aster_lens_catalog for profile:
Default_DB_Profile
Host:192.168.1.1
db:beehive
username:db_superuser
password:*****
catalog:aster_lens_catalog
Catalog table Created.

All catalog tables created.
```

Adding Entries to Visualization Catalog Tables

Whenever you create a new Visualization Table, you must add an entry for that table to the Visualization Catalog Table residing in the same database. If not, you cannot see the new visualizations in Aster Lens.

In ACT, to add a Visualization Table entry to a Visualization Catalog Table, use the INSERT command:

```
=> insert into <visualization_catalog_table_name> values
(<unique_id>, '<schema_name>', '<visualization_table_name>', '<category_name>');
```

Table 15 - 368: Visualization Catalog Table Parameters

Parameter	Description
<unique_id>	ID. Must be unique.
<schema_name>	Name of the schema with which the Visualization Table is associated.
<visualization_table_name>	Name of the Visualization Table.
<category_name>	The name of the visualization category. This name appears in the sidebar of Aster Lens. All visualizations in the Visualization Table appear under that category.

Example: Adding Entries to a Visualization Catalog Table

This example shows you how to create new visualizations using NpathViz and how to update the Visualization Catalog Table so that Aster Lens can show the visualizations in the GUI:

- 1 Using ACT, log in to an Aster Database cluster specified in the asterlens.properties file.

```
# act -U db_superuser
```

- 2 Build the table aster_lens.npathviz_input, which contains nPath output.

```
=> drop table if exists aster_lens.npathviz_input;
create table aster_lens.npathviz_input(hit_time_gmt time, ip varchar, visid_low bigint,
```

Aster Lens

Installing and Setting Up Aster Lens

```
visid_high bigint, visit_num bigint, geo_region varchar, geo_country varchar, cnt  
bigint, path varchar) distribute by hash(geo_country);  
  
=> insert into aster_lens.npathviz_input values  
('2012-03-01 23:04:11','72.152.194.83',2738316554,1217970771,1,'ga','usa',1,  
'[Cart : Billing and Shipping, Cart : Billing and Shipping,Cart]'),  
('2012-03-01 17:51:27','69.244.168.222',6917530824015025988,2857539433292760267,10,  
'mi','chn',1,['Cart : Billing and Shipping, Cart : Billing and Shipping, International']),  
('2012-03-02 17:01:21','166.205.9.10',2059641673,2798455050,11,'tx','usa',1,['Cart :  
Billing and Shipping, SEARCH, Home Page']);
```

- 3 Create the Visualization Table aster_lens.npathviz_output_table and run NpathViz on the input data stored in the aster_lens.npathviz_input table.

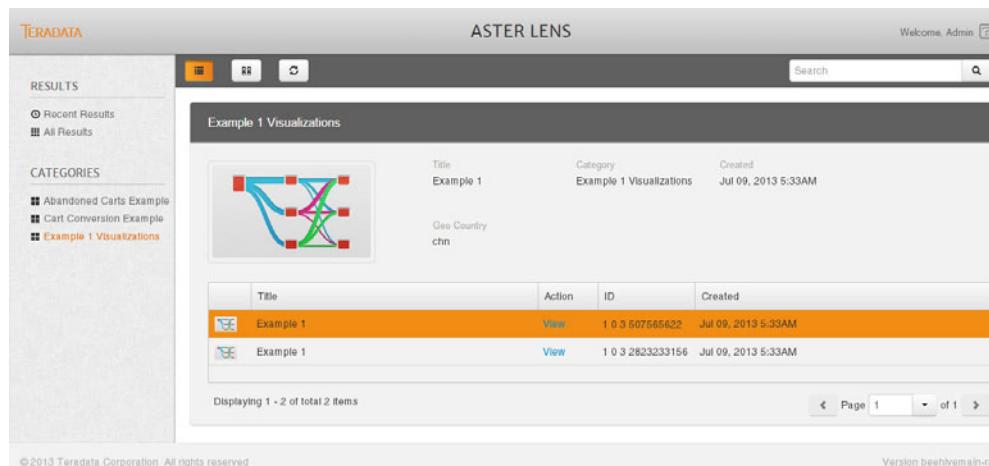
```
=> create table aster_lens.npathviz_output_table distribute by hash(geo_country) as (  
select * FROM NpathViz(  
    ON aster_lens.npathviz_input  
    PARTITION BY geo_country  
    TITLE('Example 1')  
    frequency_col('cnt')  
    path_col('path')  
    accumulate('geo_country')));  
=> insert into aster_lens_catalog values (3,'aster_lens','npathviz_output_table',  
'Example 1 Visualizations');
```

- 4 In Aster Lens, click the Refresh button.

The new category “Example 1 Visualizations” category appears in the sidebar.

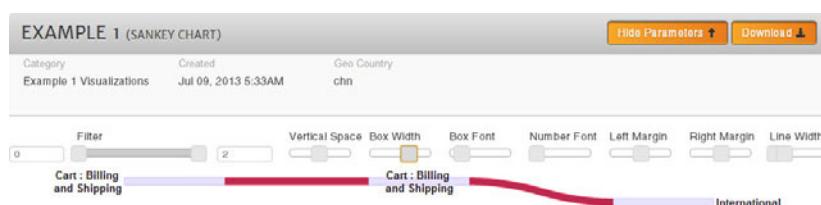
- 5 In the sidebar, click Example 1 Visualizations.

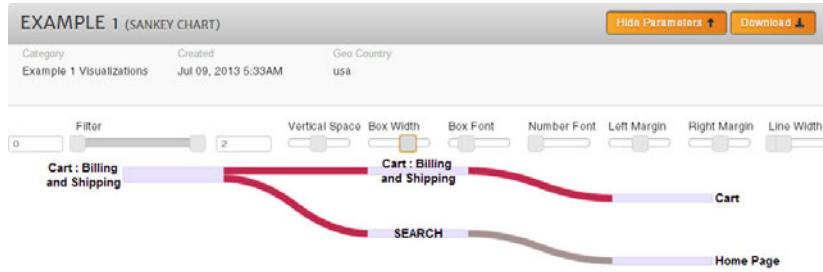
Two visualizations appear in the list.



Title	Action	ID	Created
Example 1	View	1 0 3 507566822	Jul 09, 2013 5:33AM
Example 1	View	1 0 3 2823233156	Jul 09, 2013 5:33AM

- 6 Click these visualizations to display them.





Duplicate References to the Same Visualization Table

If a Visualization Catalog Table has multiple entries that reference the same Visualization Table, Aster Lens randomly picks one of those entries and displays the category name specified by the selected entry in the sidebar.

For example, the first two rows in the example Visualization Catalog Table below reference the same Visualization Table (npathviz_output). In the sidebar in Aster Lens, either “Category 1” or “Category 2” is displayed.

Table 15 - 369: Example Visualization Catalog Table entries

id	schema_name	table_name	category_name
1	public	npathviz_output	Category 1
2	public	npathviz_output	Category 2

Removing Visualization Catalog Tables

To remove the Visualization Catalog Tables from the Aster Database clusters specified in the `AsterLens/asterlens/asterlens.properties` file, run the `AsterLens/scripts/uninstall-catalog.sh` script.

```
# sudo ./uninstall-catalog.sh
```

Installing and Uninstalling Examples

- [Installing Examples](#)
- [Uninstalling Examples](#)

Installing Examples

The Aster Lens installation script gives you the option to install the examples contained in the Aster Lens installation. However, if you choose not to install the examples and need to install them later, use the `AsterLens/examples/install-examples.sh` script.

To populate your Aster Database with the examples provided by Aster Lens:

- 1 Make sure that NpathViz and CfilterViz are installed on the Aster Database clusters that Aster Lens has access to.
- 2 If not present, create the Visualization Catalog Tables.
- 3 Run the `AsterLens/examples/install-examples.sh` script.

- 4 Log in to Aster Lens to view the visualization examples.

Uninstalling Examples

To manually uninstall examples, use the `AsterLens/examples/uninstall-examples.sh` script.

Starting and Stopping Aster Lens

- [Starting Aster Lens](#)
- [Stopping Aster Lens](#)

Starting Aster Lens

To start Aster Lens, use the `AsterLens/start-asterlens.sh` script:

```
# sudo ./start-asterlens.sh
Starting Apache ...
Starting Tomcat ...
...
```

Stopping Aster Lens

To stop Aster Lens, use the `AsterLens/stop-asterlens.sh` script:

```
# sudo ./stop-asterlens.sh
Stopping Tomcat ...
...
Stopping Apache ...
```

Uninstalling Aster Lens

The `AsterLens/uninstall-asterlens.sh` script performs the following tasks:

- Stops Aster Lens.
- If installed, removes the Aster Lens examples.
- Removes the Visualization Catalog Tables.
- Removes the symbolic link `/home/beehive/toolchain`.
- Removes the `asterlens` user and group.

To uninstall Aster Lens:

- 1 Run the `/opt/AsterLens/uninstall-asterlens.sh` script.

```
# sudo ./uninstall-asterlens.sh
```

- 2 When prompted to continue, enter y:

```
Do you want to continue? (y/n)
```

```
Y
```

```
#####
# Step 1 : Stopping Aster lens. #####
Stopping Tomcat ...
Using CATALINA_BASE:      /asterlens/AsterLens/apache-tomcat-7.0.35
Using CATALINA_HOME:       /asterlens/AsterLens/apache-tomcat-7.0.35
Using CATALINA_TMPDIR:     /asterlens/AsterLens/apache-tomcat-7.0.35/temp
Using JRE_HOME:            /asterlens/AsterLens/jdk1.6.0_39
```

```
Using CLASSPATH:          /asterlens/AsterLens/apache-tomcat-7.0.35/bin/
boots
```

```
...
```

- 3 When prompted to delete the asterlens user and group, enter **y** to delete them or **n** to keep them.

```
Do you want to delete user and group asterlens? (y/n) :
```

The `uninstall-asterlens.sh` script removes the symbolic links created in the `/etc/rc.d` folder and deletes the `/opt/AsterLens` folder and all of its contents.

Manually Installing Aster Lens

To manually install Aster Lens so that it runs on your Apache Tomcat server:

- 1 Run the `AsterLensInstaller_5-10.bin` as root:

```
sudo ./AsterLensInstaller_5-10.bin
```

This command installs the Aster Lens file in the `/opt/AsterLens` directory and starts the installation script `/opt/AsterLens/install-asterlens.sh`.



The installer adds a third-party license file (`license.html`) to the `/opt/AsterLens` directory.

-
- 2 When prompted whether to proceed with the installation, enter **n**.

```
Are you ready to proceed with the installation (y/n) :
```

```
n
```

The installation script quits.

- 3 Copy the `/opt/AsterLens/asterlens/web/webapps/asterlens.war` file into the `webapps` directory of your Apache Tomcat server.
- 4 Create the environment variable `MINIAPP_HOME` and set it to `/opt/AsterLens/asterlens`.
- 5 Modify the Apache configuration file to include these lines:

```
LoadModule rewrite_module modules/mod_rewrite.so

# Load the jk module. This is present in the toolchain
LoadModule jk_module modules/mod_jk.so

# Define the alias for the new AMC directory
Alias /asterlens ${MINIAPP_HOME}/web/webapps/asterlens

<Directory "${MINIAPP_HOME}/web/webapps/asterlens">
Options None
AllowOverride None
Order allow,deny
allow from all
</Directory>

# File containing the configuration of the local Tomcat Server
JkWorkersFile ${MINIAPP_HOME}/conf/httpd.conf.d/mod_jk_workers.properties

# Path of mod_jk log file
```

```
JkLogFile ${MINIAPP_HOME}/logs/apache_mod_jk.log

JkShmFile ${MINIAPP_HOME}/tmp/.apache_mod_jk_runtime_status

# Set the jk log level [debug/error/info]
JkLogLevel error

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

JkExtractSSL Off

# JkOptions indicate to send SSL KEY SIZE,
#JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat set the request format
# JkRequestLogFormat "%w %V %T"

JkMountCopy All

# Forward all request targeted to new AMC to Tomcat
JkMount /asterlens/* worker1

# Exceptions: Serve all static content using Apache (js, css, images)
JkUnMount /asterlens/js/* worker1
JkUnMount /asterlens/css/* worker1
JkUnMount /asterlens/img/* worker1
JkUnMount /asterlens/docs/* worker1
```

- 6 Edit the configuration file /opt/AsterLens/asterlens/asterlens.properties as described in [Configuring Aster Lens Properties](#).
- 7 Create and set up the Visualization Catalog Tables on the Aster Databases specified in the asterlens.properties file, as described in [Creating Visualization Catalog Tables](#). If you already have Visualization Tables in your databases, you can add them to their corresponding Visualization Catalog Tables as described in [Adding Entries to Visualization Catalog Tables](#).
- 8 (Optional) Install the Aster Lens examples by running the example-installation script /opt/AsterLens/examples/install-examples.sh, as described in [Installing Examples](#).
- 9 Restart your Apache HTTPD and Apache Tomcat servers.
- 10 To access Aster Lens, open a supported Web browser on a client system and, in the URL field, enter the full URL of Aster Lens:

`http://<ip_address>/asterlens/portal/`

For more information on how to connect Apache HTTPD to Apache Tomcat, see http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html

Installing Aster Lens on the Queen



Installing Aster Lens on the Queen is not recommended and is not supported. This procedure is only included for testing or demo purposes.

To install Aster Lens on the Queen:

- 1 Copy the Aster Lens installation package (`AsterLensInstaller_5-10.bin`) to the Queen.

- 2 Make the installer executable.

```
# sudo chmod +x AsterLensInstaller_5-10.bin
```

- 3 Run the installer as root:

```
# sudo ./AsterLensInstaller_5-10.bin
```

This command installs the Aster Lens files in the `/opt/AsterLens` directory and starts the installation script `install-asterlens.sh`.



The installer adds a third-party license file (`license.html`) to the `/opt/AsterLens` directory.

- 4 When prompted whether to proceed with the installation, enter `n`.

Are you ready to proceed with the installation (y/n) :

`n`

The installation script quits.

- 5 Edit the configuration file `/opt/AsterLens/asterlens/asterlens.properties` as described in [Configuring Aster Lens Properties](#).

- 6 Modify the `/home/beehive/apache/conf/conf.d/mod_jk_workers.properties` file:

- a Add `worker2` to `worker.list`.

- b Add `worker2` properties (port, host, and type) and change the port number for `worker2` by adding a 1000 to the port number used by `worker1`).

The file should look like this:

```
#  
# Copyright (c) 2011-2012 by Teradata Corporation. All rights reserved.  
#  
# Copyright (c) 2010-2011 Aster Data Systems, Inc. All rights reserved.  
#  
#  
# Apache Java Connector(mod_jk) worker configuration file  
#  
# This defines the Tomcat instances where the new AMC requests should be  
# forwarded. Note that the term "worker" here is specific to mod_jk and  
# does not refer to the nCluster worker node  
  
worker.list=worker1,worker2
```

```
worker.worker1.port=8009
worker.worker1.host=localhost
worker.worker1.type=ajp13
```

```
worker.worker2.port=9009
worker.worker2.host=localhost
worker.worker2.type=ajp13
```

7 Modify the /home/beehive/apache/conf/conf.d/jk.conf file by adding the lines in bold below:

```
# Define the alias for the new AMC directory
Alias /asterlens /opt/AsterLens/asterlens/web/webapps/asterlens
```

```
<Directory "/opt/AsterLens/asterlens/web/webapps/asterlens">
    Options None
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
```

```
# Forward all request targeted to new AMC to Tomcat
JkMount /asterlens/* worker2
```

```
# Exceptions: Serve all static content using Apache (js, css, images)
JkUnMount /asterlens/js/* worker2
JkUnMount /asterlens/css/* worker2
JkUnMount /asterlens/img/* worker2
JkUnMount /asterlens/docs/* worker2
```

8 Modify the /opt/AsterLens/asterlens/web/conf/server.xml file by adding 1000 to all port numbers specified in the file.

```
...
<Server port="9005" shutdown="SHUTDOWN">
    <!-- Security listener. Documentation at /docs/config/listeners.html
    <Listener className="org.apache.catalina.security.SecurityListener" />
    -->
...
<!-- A "Connector" represents an endpoint by which requests are received
     and responses are returned. Documentation at :
     Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
     Java AJP  Connector: /docs/config/ajp.html
     APR (HTTP/AJP) Connector: /docs/apr.html
     Define a non-SSL HTTP/1.1 Connector on port 9080
    -->
    <Connector port="9080" protocol="HTTP/1.1"
...
    <Connector port="9080" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="9443" />
...
    <Connector executor="tomcatThreadPool"
        port="9080" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="9443" />
...
    <Connector port="9443" protocol="HTTP/1.1" SSLEnabled="true"
...

```

```
<Connector port="9009" protocol="AJP/1.3" redirectPort="9443" />
...
```

- 9** In the `/home/beehive/config/procmgmtConfigs/coordinator.cfg` file, add the `MINIAPP_HOME` environment variable definition (`MINIAPP_HOME=/opt/AsterLens/asterlens`) to the `executableArgs` argument of the Tomcat job. For example:

```
{
    "jobUser": "beehive",
    "jobName": "Tomcat on REPLACE_NODE_IP",
    "Tasks": [
        {
            "taskName": "Tomcat",
            "nodeIps": "REPLACE_NODE_IP",
            "executableLocation": "/bin/bash",
            "executableArgs": "-c, JAVA_HOME=/home/beehive/toolchain/x86_64-unknown-linux-gnu/jdk1.6.0_39 CATALINA_BASE=/home/beehive/amc/webserver MINIAPP_HOME=/opt/AsterLens/asterlens CATALINA_OPTS=-XmxJAVA_HEAP_LIMIT /home/beehive/toolchain/noarch/apache-tomcat-7.0.35/bin/catalina.sh run",
            "maxTaskRestarts": -1
        }
    ]
},
```

- 10** In the `/opt/AsterLens/start-asterlens.sh` script, comment these lines out:

```
...
#echo "Starting Apache ..."
#LD_LIBRARY_PATH=$LD_LIBRARY_PATH TOP_DIR=$TOP_DIR
MINIAPP_HOME=$MINIAPP_HOME $HTTPD_HOME/bin/httpd -f $MINIAPP_HOME/conf/httpd.conf -k start
#if [ $? = "0" ]
#then
#    echo "Starting Tomcat ..."
#    JAVA_HOME=$JAVA_HOME JRE_HOME=$JAVA_HOME
#    CATALINA_BASE=$MINIAPP_HOME/web CATALINA_OPTS=-Xmx254m
#    MINIAPP_HOME=$MINIAPP_HOME $CATALINA_HOME/bin/catalina.sh run >> $MINIAPP_LOG 2>&1 &

if [ $? = "0" ]
then
    hostname=$(hostname)
    echo "Aster Lens is now running on '$hostname'. Point your browser to this host to view Aster Lens"
else
    echo " "
    echo "Failed to start tomcat."
    exit 1
fi
#else
#    echo " "
```

```
#      echo "Failed to start apache."
#      exit 1
#fi
```

- 11 Create and set up the Visualization Catalog Tables on the Aster Databases, as described in [Creating Visualization Catalog Tables](#). If you already have Visualization Tables in your databases, you can add them to their corresponding Visualization Catalog Tables as described in [Adding Entries to Visualization Catalog Tables](#).

- 12 (Optional) Install the Aster Lens examples by running the example installation script, as described in [Installing Examples](#).

- 13 Restart the Queen:

```
# /etc/init.d/local restart
Shutting down nCluster Services: [ OK ]
Starting nCluster Services: [ OK ]
```

- 14 Activate the cluster using the AMC.

- 15 Run the /opt/AsterLens/start-asterlens.sh script.

- 16 To access Aster Lens, open a supported Web browser on a client system and, in the URL field, enter the full URL of Aster Lens:

```
http://<ip_address>/asterlens/portal/
```

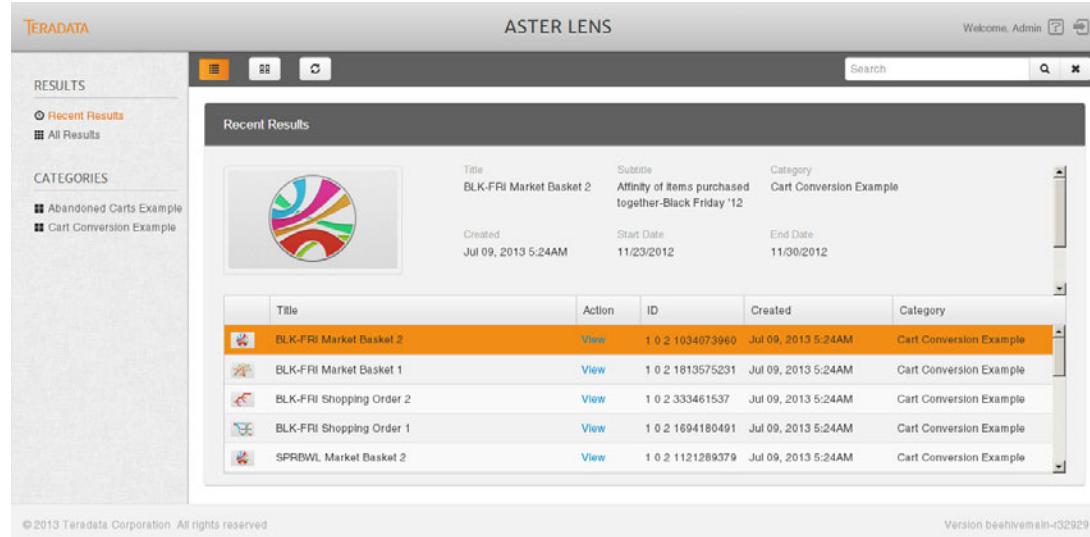
Using Aster Lens

- [Displaying Visualizations in Table View](#)
- [Displaying the Sidebar](#)
- [Displaying Recent Results](#)
- [Using the Search Field](#)
- [Logging Out](#)
- [Refreshing the Server Cache](#)
- [Displaying Summary Information](#)
- [Viewing a Visualization](#)
- [Downloading Visualizations](#)
- [Displaying Visualizations By Category](#)
- [Viewing Tree Visualizations](#)
- [Viewing Chord Visualizations](#)
- [Viewing Sankey Visualizations](#)
- [Viewing Sigma Visualizations](#)
- [Viewing GEXF and Graphviz Visualizations](#)
- [Viewing Log Files](#)

Displaying Visualizations in Table View

To display visualizations in **Table View**, click the **Show Table View** button. Aster Lens displays a list of the visualization belonging to the same category as the first visualization in **Card View**.

The first visualization in the list is highlighted and summary information about it is displayed above the table.



The screenshot shows the Aster Lens application window. On the left, there's a sidebar with 'RESULTS' (Recent Results selected), 'CATEGORIES' (Abandoned Carts Example, Cart Conversion Example), and other navigation links like 'TERADATA' and 'ASTER LENS'. The main area is titled 'Recent Results' and displays a table of visualizations. At the top of the table, there's a summary row for 'BLK-FRI Market Basket 2'. Below this is a table with columns: Title, Action, ID, Created, and Category. The table contains five rows, each with a visualization icon, a 'View' link, an ID, a creation date, and a category. The first row ('BLK-FRI Market Basket 2') is highlighted with a yellow background.

Title	Action	ID	Created	Category
BLK-FRI Market Basket 2	View	1 0 2 1034073960	Jul 09, 2013 5:24AM	Cart Conversion Example
BLK-FRI Market Basket 1	View	1 0 2 1813575231	Jul 09, 2013 5:24AM	Cart Conversion Example
BLK-FRI Shopping Order 2	View	1 0 2 333461537	Jul 09, 2013 5:24AM	Cart Conversion Example
BLK-FRI Shopping Order 1	View	1 0 2 1694180491	Jul 09, 2013 5:24AM	Cart Conversion Example
SPRBWL Market Basket 2	View	1 0 2 1121289379	Jul 09, 2013 5:24AM	Cart Conversion Example

To view a visualization, click **View** or the visualization icon.

Displaying the Sidebar

The Aster Lens sidebar lists all the report categories of the visualizations that Aster Lens has access to.

To display (or hide) the sidebar, click the **Toggle Sidebar** button.

Displaying Recent Results

To display recent results:

- 1 Display the sidebar if necessary.
- 2 Click **Recent Results**.

Aster Lens displays the most recent results.

Using the Search Field

Use the **Search** field to find visualizations. In the **Search** field, enter one or more keywords that identify the visualizations that you are looking for. The visualizations that match the search criteria are displayed under **Search Results**.

The screenshot shows the Aster Lens application window. At the top, there's a navigation bar with the Teradata logo, the title 'ASTER LENS', and a user 'Welcome, Admin'. Below the title is a search bar containing 'Basket' with a magnifying glass icon. On the left, there's a sidebar with sections for 'RESULTS' (Recent Results, All Results) and 'CATEGORIES' (Abandoned Carts Example, Cart Conversion Example). The main area is titled 'Search Results' and displays a single visualization card for 'BLK-FRI Market Basket 2'. This card includes the title, subtitle ('Affinity of items purchased together-Black Friday '12'), category ('Cart Conversion Example'), creation date ('Jul 09, 2013 5:24AM'), start date ('11/23/2012'), and end date ('11/30/2012'). Below the card is a table listing 16 items, each with a small thumbnail, title ('BLK-FRI Market Basket 2', 'BLK-FRI Market Basket 1', 'SPRBWL Market Basket 2', 'SPRBWL Market Basket 1'), action ('View'), ID ('1 0 2 1034073960', '1 0 2 1813575231', '1 0 2 1121289379', '1 0 2 1384057553'), created date ('Jul 09, 2013 5:24AM', 'Jul 09, 2013 5:24AM', 'Jul 09, 2013 5:24AM', 'Jul 09, 2013 5:24AM'), and category ('Cart Conversion Example', 'Cart Conversion Example', 'Cart Conversion Example', 'Cart Conversion Example'). At the bottom, it says 'Displaying 1 - 16 of total 16 items' and has a page navigation bar.

You can also search for visualizations by their create time (the **Created** field in the summary information, or the `time_stamp` column in `NpathViz` or `CfilterViz` output table).

To search by create time, use this format:

`MMM DD, YYYY`

For example:

- Jun 28, 2013
- Jul 08, 2013

You can use other formats like the ones below, but the results are not highlighted. Only exact string match are highlighted.

`DD MMM YYYY`

`YYYY MMM DD`

`DD YYYY MMM`

Refreshing the Server Cache

To refresh the server cache, click the **Refresh Server Cache** button. Aster Lens loads any visualization update to the GUI.



Tip: If you make a change to a Visualization Table or Visualization Catalog Table and click **Refresh Server Cache**, Aster Lens updates the screen accordingly. However, if other users who are not aware of the change try to access the affected visualizations, they get an error message. To refresh Aster Lens so that all visualizations are up to date, the affected users should use the **Refresh** button on their Web browsers (and not the **Refresh Server Cache** button in Aster Lens).

Logging Out

To log out, click the **Log Out** button.

Displaying Summary Information

In **Card View**, each card displays the following about a visualization:

- Title of the visualization
- The visualization category
- Icon representing the graph type
- The date when the visualization was created

To display additional summary information, click the **Information** button. The **Information** panel displays this information:

- Title of the visualization
- (If provided) Subtitle of the visualization
- (If provided) Keywords associated with the visualization
- (If provided) The visualization start and end times

This information is also available in **Table View**.

Viewing a Visualization

To view a visualization:

- In **Card View**, click the icon of the visualization.
- In **List View**, click **View** or the visualization icon.

When in **Card View**, if you choose to display all the results, Aster Lens displays 25 visualizations at most. If there are more than 25 visualizations, Aster Lens displays an empty card at the end of the list. This card contains the **Load More** link. Click this link to load additional visualizations.

Downloading Visualizations

To download a visualization:

- 1 Open the window displaying the visualization.
- 2 Click **Download**.

Aster Lens downloads the visualization as a ZIP archive into your default downloads folder. When you extract the contents of the ZIP archive, a new folder is created containing the extracted content. At the root level of that folder is an HTML file. Open this file in a web browser to display the visualization.



For the visualizations to display correctly, extract the contents of the ZIP archive first instead of opening the HTML file from within the ZIP archive.

Displaying Visualizations By Category

To display the visualizations that belong to a category:

- 1 Display the sidebar.
- 2 Click one of the categories to display the corresponding visualizations.

Viewing Tree Visualizations

When you display a tree visualization:

- Click the nodes to expand or collapse the tree.
- Hover over lines to show the path values.

Viewing Chord Visualizations

When you display a chord visualization, move your mouse over the category in the graph that you want to focus on.

Viewing Sankey Visualizations

When you display a Sankey visualization, move the mouse over the areas of the graph that you want to focus on.

Additionally, you can use these fields to control the display of the tree:

- **Filter:** Controls the display of the lines connecting nodes. By default, the graph shows all of the lines. For example, to show only the lines that have a weight number of 1, enter 1 in the two fields to the left and right of the Filter slider. To show the lines whose widths are 10 or more, but less than 56, enter 10 in the left field and 56 in the right field.
- **Vertical Space:** Controls the vertical space between nodes.
- **Box Width:** Controls the width of the node box.
- **Box Font:** Controls the size of the text in the node box.
- **Number Font:** Controls the size of the numbers that appear when you hover over a node.
- **Left Margin:** Moves the left margin of the tree.
- **Right Margin:** Moves the right margin of the tree.
- **Line Width:** Controls the width of the lines connecting the nodes.

Viewing Sigma Visualizations

Like the other graph types, the Sigma graph is also interactive:

- To only display the neighbors of a node, hover over the node.
- Use the mouse wheel to zoom in and out of the graph.
- Click and drag to pan.

Also, you can control the layout and formatting of Sigma charts.

Controlling Layout

You can control the following layout settings:

Table 15 - 370: Layout settings for Sigma

Parameter	Description
Layout Mode	Starts or stops the laying out of the graph. Whenever you change any of the layout settings, the layout counter starts the layout count down from 10 to 0. You can stop it at any time.
Layout Presets	A drop-down menu that provides layout presets to choose from.
Gravity	Controls the distance between nodes and the center of the screen. You can use this setting to bring nodes closer to the center of the screen and prevent nodes islands from drifting away.
Attraction	Brings nodes closer together in proportion to the distance between nodes.
Scaling Ratio	Controls the amount of repulsion in the graph. To make a graph more sparse, decrease the scaling ratio.
Edge Influence	Controls the influence assigned to weight of the edges.
Strong Gravity	Brings nodes even closer to the center of the screen than the Gravity setting.
LinLog Mode	Controls the tightness of node clusters.
Repulsion	Increases repulsion between nodes, which results in larger graphs.

Controlling Formatting

You can control the following formatting settings:

Table 15 - 371: Formatting settings for Sigma

Parameter	Description
Reset Zoom	Resets zooming to the default value.
Color Theme	Changes the background color of the chart.
Scale Nodes	Controls the size of the nodes.
Scale Edges	Controls the width of the lines connecting the nodes.
Label Threshold	Controls the display of labels.
Edge Type	Controls the shape of the line connecting the nodes (straight or curved).
Edge Labels	Controls the display of the line labels.

Viewing GEXF and Graphviz Visualizations

Aster Lens lists Graph Exchange XML Format (GEXF) and Graphviz visualizations in **Card View** and **Table View**, but does not display the actual visualizations. In Aster Lens, when you try

to view a GEXF or Graphviz visualization, a message appears in the display window stating that to view the graph, you should download it to your desktop. The downloaded ZIP archive contains all the files necessary to display the visualization.

Viewing Log Files

The Aster Lens log files are located in the directory `/opt/AsterLens/asterlens/logs/`. The web application log entries are in the log file `/opt/AsterLens/asterlens/logs/asterlens-tomcat.log`.

APPENDIX A List of Functions

Time Series, Path, and Attribution Analysis

Cumulative Moving Average

```
SELECT *
FROM CMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
    COLUMNS('column_names')
) ;
```

SAX

```
SELECT * FROM SAX (
    ON {table_name | view_name | (query)}
    PARTITION BY partition_column
    ORDER BY ordering_columns
    VALUE_COLUMN_NAMES('value_column1', 'value_column2', ...)
    [PAA_SIZE('paa_size_value_column1', 'paa_size_value_
    column2', ...)]
    [MAX_PAA_SIZE('max_paa_value')]
    [ALPHABET_SIZE('alphabet_size_value_column1', 'alphabet_
    size_value_column2', ...)]
    [ZNORM('znorm_value_column1', 'znorm_value_
    column2', ...)]
    [MEAN_VAL('mean_value_column1', 'mean_value_
    column2', ...)]
    [STDEV_VAL('stdev_value_column1', 'stdev_value_
    column2', ...)]
    [ACCUMULATE('accumulate_column1', 'accumulate_
    column2', ...)])
    [TIME_COLUMN_NAME('time_column1', 'time_column2', ...)]
);
SELECT * FROM SAX(
    ON {table_name | view_name | (query)} as input
    PARTITION BY partition_columns
    ORDER BY ordering_columns
    ON <(SELECT partition_column1, partition_column2, ...
    mean_value1 AS value_column1, mean_value2 AS value_
    column2, ... FROM Statistics_Table)> as meanstats
    PARTITION BY partition_columns
    ON <(SELECT partition_column1, partition_column2, ...
    stdev_value1 AS value_column1, stdev_value2 AS value_
    column2, ... FROM Statistics_Table)> AS stdevstats
    PARTITION BY partition_columns
    VALUE_COLUMN_NAMES('value_column1', 'value_column2', ...)
```

```

[PAA_INTERVAL('paa_interval_value_column1', 'paa_
interval_value_column2', ...)]
[ALPHABET_SIZE('alphabet_size_value_column1', 'alphabet_-
size_value_column2', ...)]
[ZNORM('znorm_value_column1', 'znorm_value_-
column2', ...)]
[MEAN_VAL('mean_value_column1', 'mean_value_-
column2', ...)]
[STDEV_VAL('stdev_value_column1', 'stdev_value_-
column2', ...)]
[ACCUMULATE('accumulate_column1', 'accumulate_-
column2', ...)]
[TIME_COLUMN_NAME('time_column1', 'time_column2', ...)]
);

```

Path Generator

```

SELECT *
FROM path_generator
(
    ON {table_name | view_name | (query)}
    SEQ('sequence_column')
    [DELIMITER('delimiter_character')]
);

```

Path Starter

```

SELECT *
FROM PATH_START
(
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    CNT('count-column')
    [DELIMITER(',')]
    PARENT('parent-column')
    PARTITIONNAMES('partitionby-col-name' [ , ... ] )
    NODE('node-column')
);

```

Path Summarizer

```

SELECT *
FROM PATH_SUMMARIZER
(
    ON {table_name|view_name|(query)}
    PARTITION BY expression [, ...]
    CNT('count_column')
    DELIMITER(',')
    SEQ('sequence-column')
    PARTITIONNAMES('partitionby-col-name' [ , ... ] )
    HASH('true|false')
    PREFIX('prefix-column')
);

```

Sessionization

```

SELECT *
FROM SESSIONIZE(
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    ORDER BY order_by_columns
    TIMECOLUMN ('timestamp_column')
    TIMEOUT (session_timeout_value)
    [RAPIDFIRE (min_human_click_lag)]
    [EMITNULL]
);

```

Attribution

Attribution (Multiple Input)

```

SELECT * FROM attribution
(
    ON {input_table | view | query}
    AS input PARTITION BY user_id
    ORDER BY time_stamp

    ON conversion_event_table AS conversion DIMENSION
    ON excluding_event_table AS excluding DIMENSION
    ON optional_event_table AS optional DIMENSION

    ON model1_table AS model1 DIMENSION
    ON model2_table AS model2 DIMENSION

    EVENT_COLUMN_NAME('event_column')
    TIMESTAMP_COLUMN_NAME('timestamp_column')
    WINDOW('rows:K | seconds:K | rows:K&seconds:K2')

) ORDER BY user_id, time_stamp;

```

Attribution (Single Input)

```

SELECT * FROM attribution
(
    ON {input_table | view | query}
    PARTITION BY expression [, ...]
    ORDER BY order_by_columns
    EVENT_COLUMN_NAME('event_column')
    CONVERSION_EVENT_TYPE_VALUE('click1', 'click2', ...)
    [EXCLUDING_EVENT_TYPE_VALUE('email')]
    [OPTIONAL_EVENT_TYPE_VALUE('optional1', 'optional2')]
    TIMESTAMP_COLUMN_NAME('timestamp_column')
    WINDOW('rows:K | seconds:K | rows:K&seconds:K')
    MODEL1('TYPE', 'K|EVENT:WEIGHT:MODEL:PARAMETERS', ...)
    [MODEL2('TYPE', 'K|EVENT:WEIGHT:MODEL:PARAMETERS', ...)]
);

```

FrequentPaths

```

SELECT *
FROM FrequentPaths(
    ON (SELECT 1)

```

```

PARTITION BY 1
[DOMAIN('host_ip')]
[DATABASE('db_name')]
[USERID('user_id')]
PASSWORD('password')
INPUTTABLE('input_table_name')
PARTITIONCOLUMNS('partition_column_list')
[TIMECOLUMN('sort_column_name')]
[PATHFILTERS('filter_list')]
[GROUPBYCOLUMNS(groupby_column_list)]
[SEQUENCEPATTERNRELATION('sequence_pattern_table_name')]
[ITEMCOLUMN('sequence_column_name')]
[ITEMDEFINITION('item_definition_table:
    [id_column:definition_column:item_column]')]
[PATHCOLUMN('path_column_name')]
MIN SUPPORT('minimum_support_value')
[MAXLENGTH('maximum_length')]
[MINLENGTH('minimum_length')]
);

```

DWT

```

SELECT * FROM dwt(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    METATABLE('meta_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
    [PARTITIONCOLUMNS('partition_column_name1',
        'partition_column_name2', ...,
        'partition_column_nameN')]
    WAVELETNAME('wavelet_name') |
        WAVELETFILTERTABLE('wavelet_filter_table_name')
    LEVEL(level)
    [EXTENSIONMODE('<extension mode>')]
);

```

IDWT

```

SELECT * FROM idwt(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    METATABLE('meta_table_name')
    OUTPUTTABLE('output_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
    [PARTITIONCOLUMNS('partition_column_name1',

```

```

    'partition_column_name2', ...,
    'partition_column_nameN') ]
) ;

```

DWT2d

```

SELECT * FROM dwt2d(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    METATABL('meta_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    [PARTITIONCOLUMNS('partition_column_name1',
        'partition_column_name2', ..., 'partition_column_na-
        meN') ]
    INDEXCOLUMNS('indexy_column_name', 'indexx_column_name')
    [RANGE('starty,startx), (endy, endx)'])
    [WAVELETNAME('wavelet_name') | |
    WAVELETFILERTABLE('wavelet_filter_table_name')]
    LEVEL(level)
    [EXTENSIONMODE('extension_mode')]
    [COMPACTOUTPUT('true' | 'false')]
) ;

```

IDWT2d

```

SELECT * FROM idwt2d(
    ON (SELECT 1) PARTITION BY 1
    [DOMAIN( 'host_ip' )]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    METATABL('meta_table_name')
    OUTPUTTABLE('output_table_name')
    INPUTCOLUMNS('col1', 'col2', ..., 'colN')
    SORTCOLUMN('sort_column_name')
    [PARTITIONCOLUMNS('partition_column_name1',
        'partition_column_name2', ...,
        'partition_column_nameN') ]
    [COMACTOUTPUT('true' | 'false')]
) ;

```

Pattern Matching with Teradata Aster nPath

nPath

```

SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
FROM NPATH

```

```

( on_clause1 [ on_clause2 ... on_clausen ]
  MODE ( OVERLAPPING | NONOVERLAPPING )
  PATTERN ( 'pattern_of_symbols' )
  SYMBOLS ( symbol_predicate AS symbol [, ...] )
  RESULT ( aggregate_function( expression OF symbol ) AS
alias [, ...] )
)
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST |
LAST } ] [, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ];

```

Statistical Analysis

Approximate Distinct Count (count_approx_distinct)

```

SELECT *
  FROM APPROXDCOUNTREDUCE
  (
    ON
      (SELECT *
        FROM APPROXDCOUNTMAP
        (
          ON {table_name | view_name | (query)}
          COLUMNS ('column_name')
          [ERROR ('error_tolerance')])
        )
      )
    PARTITION BY expression [, ...]
  );
SELECT *
  FROM APPROXDCOUNTREDUCE
  (
    ON
      ( APPROXDCOUNTMAP
        (
          ON {table_name | view_name | (query)}
          COLUMNS ('column_name')
          [ERROR ('error_tolerance')])
        )
      )
    PARTITION BY expression [, ...]
  );

```

Approximate Percentile (approx_percentile)

```

SELECT *
  FROM approxPercentileReduce
  (

```

```

    ON (
        SELECT *
        FROM approxPercentileMap
        (
            ON {table_name | view_name | (query ) }
            TARGET_COLUMN( 'column_name' )
            ERROR( tolerance_value )
            [GROUP_COLUMNS('column_name' [ , ...] ) ]
        )
    )
PARTITION BY expression [, ...]
PERCENTILE(percentile [, ... ] )
[GROUP_COLUMNS('column_name' [ , ...] ) ] );

```

Correlation (stats correlation)

```

SELECT *
FROM CORR_REDUCE
(
    ON CORR_MAP
    (
        ON {table_name | view_name | ( query ) }
        COLUMNPAIRS ( 'col1:col2','col2:col3',... )
        KEY_NAME ( 'key_name' )
    )
    PARTITION BY key_name
);

```

Histogram

```

SELECT *
FROM histogram_map
(
    ON {table_name | view_name | query }
    VALUE_COLUMN (column_name)
    [ BIN_SIZE ( size of the equi-sized bins) ]
    [ START_VALUE( start value of the first bin) ]
    [ INTERVALS( [<min>:<max>], ... ) ]
    [ BIN_COLUMN_NAME( output-column-name ) ]
    [ START_COLUMN_NAME( output-column-name ) ]
    [ END_COLUMN_NAME( output-column-name ) ]
    [ FREQUENCY_COLUMN_NAME ( output-column-name ) ]
);

```

Enhanced Histogram Functions

hist_map

Syntax 1a

```

SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON (select count(*) as stat_count,
        sum(value_col) as stat_sum,
        sum(value_col*value_col) as stat_squared_sum,
        min(value_col) as stat_min,

```

```

        max(value_col)      as stat_max
    from data_table where value_col is not null and
    not value_col = 'infinity' and
    not value_col = '-infinity')
) as data_stat DIMENSION
BIN_SELECT('Sturges'|'Scott'|number_of_bins)
VALUE_COLUMN('value_col')
[INCLUSION('left'||'right'))];

```

Syntax 1b

```

SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON hist_prep(
        ON data_table VALUE_COLUMN('value_col'))
        as data_stat DIMENSION
    BIN_SELECT('Sturges'|'Scott'|number_of_bins)
    VALUE_COLUMN('value_col')
    [INCLUSION('left'||'right'))];

```

Syntax 2a

```

SELECT * FROM hist_map(
    ON data_table PARTITION BY ANY
    ON bin_breaks DIMENSION
    [INCLUSION('left'||'right'))]
    VALUE_COLUMN('value_col')
    BREAKS_COLUMN('breaks_col')
);

```

Syntax 3

```

SELECT * FROM hist_map(
    ON data_table
    STARTVALUE('bin_start')
    BINSIZE('bin_size')
    ENDVALUE('bin_end')
    [INCLUSION('left'||'right'))]
    VALUE_COLUMN('value_col')
);

```

hist_reduce

```

SELECT * FROM hist_reduce(
    ON hist_map(...) PARTITION BY 1
);

```

Linear Regression (stats linear reg)

```

SELECT *
FROM LINREG
(
    ON LINREGMATRIX
    (
        ON {table_name | view_name | (query)}
    )
    PARTITION BY 1
);

```

Generalized Linear Model (stats glm)

```
SELECT *
FROM GLM (
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    [PASSWORD('password')]
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    [COLUMNNAMES('column_names')]
    [CATEGORICALCOLUMNS('columnname_value_pair' [,...])]
    [FAMILY('family')]
    [LINK('link')]
    [WEIGHT('weight_column_name')]
    [THRESHOLD('threshold')]
    [MAXITERNUM('max_iterations')]
) ;
```

Generalized Linear Model Prediction (stats glmpredict)

```
SELECT * FROM GLMPREDICT(
    ON input_table
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    [PASSWORD('password')]
    INPUTTABLE('input_table_name')
    MODELTABLE('model_table_name')
    ACCUMULATE('column_names')
    FAMILY('family')
    LINK('link')
) ;
```

Principal Component Analysis (PCA)

```
SELECT * FROM pca_reduce (
    ON pca_map
    (
        ON target_table
        [TARGET_COLUMNS(target_columns)]
    )
    PARTITION BY 1
    [COMPONENTS(num_components)]
) ORDER BY component_rank;
```

Simple Moving Average (stats smavg)

```
SELECT *
FROM SMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
```

```

    COLUMNS('column_names')
    RETURN_ALL('true|false')
    WINDOW_SIZE('window_size')
) ;

```

Weighted Moving Average (stats wmavg)

```

SELECT *
FROM WMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
    COLUMNS('column_names')
    RETURN_ALL('true|false')
    WINDOW_SIZE('window_size')
) ;

```

Exponential Moving Average (stats emavg)

```

SELECT *
FROM EMAVG
(
    ON {table_name|view_name|(query)}
    PARTITION BY partition_column
    ORDER BY order_by_column
    [COLUMNS('column_names')]
    [RETURN_ALL('true|false')]
    [START_ROWS('number')]
    [ALPHA('alpha_value')]
) ;

```

Volume-Weighted Average Price (stats vwap)

```

SELECT *
FROM VWAP
(
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    ORDER BY date_column
    [PRICE('price_column')]
    [VOLUME ('volume_column')]
    [TIMEINTERVAL('number_of_seconds')]
    [DT('date_column')]
) ;

```

K-Nearest Neighbor Classification Algorithm (knn)

```

SELECT * FROM knn(
    ON (SELECT 1)
    PARTITION BY 1
    TRAINING_TABLE('training_table_name')
    TEST_TABLE('test_table_name')
    NUMBERK(k)
    RESPONSECOLUMN('response_column')
    TEST_POINT_KEY('test_id_column')
)

```

```

DISTANCE_FEATURES('column1' [, ...])
[ VOTING_WEIGHT(voting_weight) ]
[ OUTPUT_TABLE('output_table_name') ]
[ DOMAIN('host_ip') ]
[ DATABASE('database_name') ]
[ USERID('user_name') ]
PASSWORD('password')
[CUSTOMIZED_DISTANCE('jar_name', 'distance_class_name')]
[FORCE_MAPREDUCE('force_mapreduce') ]
[PARTITION_BLOCK_SIZE('partition_Block_Size')] );

```

svm

```

CREATE TABLE output_table (PARTITION KEY(vec_index)) AS
SELECT vec_index, avg(vec_value) as vec_value
FROM svm(
    ON input_table
    PARTITION BY id_column
    OUTCOME( outcome_column )
    ATTRIBUTE_NAME( attr_name_column )
    ATTRIBUTE_VALUE( attr_value_column )
)
GROUP BY vec_index;

```

svm_predict

```

SELECT * FROM svm_predict(
    ON input_table
    PARTITION BY id_column
    ATTRIBUTE_NAME(attr_name_column)
    ATTRIBUTE_VALUE(attr_value_column)
    THRESHOLD('threshold_value1', ..., 'threshold_valueN')
    WEIGHT_FILE(weight_file_name)
);

```

ConfusionMatrix

```

select * from ConfusionMatrix(
    ON {table_name|view_name|(query)} PARTITION BY <expect_column_name>
    EXPECTCOLUMN('expect_column_name')
    PREDICTCOLUMN('result_column_name')
);

```

Percentile

```

SELECT * FROM percentile(
    ON <table>
    PARTITION BY <PARTITION BY col> [, <PARTITION BY col>]*
    PERCENTILE('<percentile>' [, '<percentile>']*)
    TARGET_COLUMNS('<column_name>' [, '<column_name>']*)
    [GROUP_COLUMNS('<column_name>' [, '<column_name>']*)]
);

```

distrnmatch (“Hypothesis Test” Mode)

Syntax (Continuous Distributions) (version 1.0)

Option 1: FACT + DIMENSION

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT RANK() OVER (PARTITION BY col1,
            col2, ..., colN ORDER BY column_name) AS rank, *
            FROM input_table WHERE column_name IS NOT NULL)
        AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size FROM input_table WHERE
            column_name IS NOT NULL GROUP BY col1, col2, ...,
            colN) AS groupstats DIMENSION
        VALUECOLUMN(column_name)
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellsize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Option 2: CO-GROUP

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT RANK() OVER (PARTITION BY col1, col2, ...,
            colN ORDER BY column_name) AS rank,
            * FROM input_table
            WHERE column_name IS NOT NULL) AS input PARTITION BY
            col1, col2, ..., colN
        ON (SELECT col1, col2, ..., colN, COUNT(*) AS
            group_size FROM input_table WHERE
            column_name IS NOT NULL GROUP BY col1, col2, ...,
            colN) AS groupstats PARTITION BY col1, col2, ...,
            colN
        VALUECOLUMN( column_name )
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
    )
    PARTITION BY col1, col2, ..., colN
);
```

Syntax (Discrete Distributions) (version 1.0)

Option 1: FACT + DIMENSION

```
SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleinput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1)) OVER
            (PARTITION BY col1, col2, ..., colN ORDER BY
            column_name) AS rank, col1, col2, ..., colN,
            column_name FROM input_table WHERE column_name IS
```

```

NOT NULL GROUP BY col1, col2, ..., colN, column_name) AS input
PARTITION BY ANY
    ON (SELECT col1, col2, ..., colN, COUNT(*) AS group_size
        FROM input_table WHERE column_name IS NOT NULL GROUP BY col1, col2, ..., colN) AS groupstats DIMENSION
        VALUECOLUMN( column_name )
        [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
        DISTRIBUTIONS('distribution1:parameter1', ... )
        [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
        [ MINGROUPSIZE( minGroupSize ) ]
        [ CELLSIZE( cellSize ) ]
)
PARTITION BY col1, col2, ..., colN
);

```

Option 2: CO-GROUP w/ ORDER BY

```

SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts, SUM(COUNT(1)) OVER
            (PARTITION BY col1, col2, ..., colN
                ORDER BY column_name) AS rank, col1,
                col2, ..., colN, column_name
            FROM input_table WHERE column_name IS NOT NULL
            GROUP BY col1, col2, ..., colN, column_name)
            AS input PARTITION BY col1, col2, ..., colN
                ORDER BY column_name
        ON (SELECT col1, col2, ..., colN, COUNT(*) AS group_size
            FROM input_table WHERE column_name IS NOT NULL GROUP
                BY col1, col2, ..., colN) AS groupstats PARTITION BY
                    col1, col2, ..., colN
                    VALUECOLUMN( column_name )
                    [ TESTS('KS', 'CvM', 'AD', 'CHISQ') ]
                    DISTRIBUTIONS('distribution1:parameter1', ... )
                    [ GROUPINGCOLUMNS( col1, col2, ..., colN ) ]
                    [ MINGROUPSIZE( minGroupSize ) ]
                    [ CELLSIZE( cellSize ) ]
)
PARTITION BY col1, col2, ..., colN
);

```

distnmatch (“Best Match” Mode)

Syntax for real type input data (version 1.0)

```

SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT RANK() OVER (PARTITION BY col1, col2, ..., colN
            ORDER BY column_name) AS rank, *
            FROM input_table WHERE column_name IS NOT NULL)
            AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size,
            AVG(column_name) AS mean,
            STDDEV(column_name) AS sd,
            CASE

```

```

        WHEN MIN(column_name) > 0 THEN
            VG(LN(CASE WHEN column_name > 0
                       THEN column_name ELSE 1 END))
        ELSE 0
    END AS mean_of_ln,
    CASE
        WHEN MIN(column_name) > 0 THEN
            STDDEV(LN(CASE WHEN column_name > 0 THEN
                           column_name ELSE 1 END))
        ELSE -1
    END AS sd_of_ln,
    MAX(column_name) AS maximum, MIN(column_name)
                           AS minimum
FROM input_table
WHERE column_name IS NOT NULL GROUP BY col1, col2, ..., colN
AS groupstats DIMENSION
VALUECOLUMN(column_name)
[TESTS('KS', 'AD', 'CHISQ')]
[DISTRIBUTIONS('distribution1:parameter1', ...)]
[GROUPINGCOLUMNS(col1, col2, ..., colN)]
[MINGROUPSIZE(minGroupSize)]
[CELLSIZE(cellszie)]
)
PARTITION BY col1, col2, ..., colN
[TOP('top')]);

```

Syntax for integer type input data (version 1.0)

```

SELECT * FROM DistnmatchReduce (
    ON DistnmatchMultipleInput (
        ON (SELECT COUNT(1) AS counts,
            SUM(COUNT(1)) OVER (PARTITION BY col1, col2, ..., colN
                ORDER BY column_name) AS rank,
            col1, col2, ..., colN, column_name
        FROM input_table
        WHERE column_name IS NOT NULL
        GROUP BY col1, col2, ..., colN, column_name)
        AS input PARTITION BY ANY
        ON (SELECT col1, col2, ..., colN,
            COUNT(*) AS group_size,
            AVG(column_name) AS mean,
            STDDEV(column_name) AS sd,
            MAX(column_name) AS maximum,
            MIN(column_name) AS minimum
        FROM input_table
        WHERE column_name IS NOT NULL
        GROUP BY col1, col2, ..., colN)
        AS groupstats DIMENSION
        VALUECOLUMN(column_name)
        [TESTS('KS', 'AD', 'CHISQ')]
        [DISTRIBUTIONS('distribution1:parameter1', ...)]
        [GROUPINGCOLUMNS(col1, col2, ..., colN)]
        [MINGROUPSIZE(minGroupSize)]
        [CELLSIZE(cellszie)]
    )
    PARTITION BY col1, col2, ..., colN
)

```

```

        [TOP('top')]
);

lars

SELECT * FROM LARS (
    ON (SELECT 1)
    PARTITION BY 1
    [ DOMAIN( 'host_ip' ) ]
    [ DATABASE('db_name') ]
    [ USERID('user_id') ]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    COLUMNNAMES('Y', 'X1', 'X2', ..., 'Xp')
    [ METHOD('lar' | 'lasso') ]
    [ INTERCEPT('true' | 'false') ]
    [ NORMALIZE('true' | 'false') ]
    [ MAXITERNUM('max_iterations') ]
);
;

SELECT * FROM output_table_name WHERE steps <> 0 ORDER BY
steps;

```

larspredict

```

SELECT * FROM LARSPREDICT (
    ON input_table AS data PARTITION BY ANY
    ON model_table AS model DIMENSION
    [MODE('STEP | FRACTION | NORM | LAMBDA') ]
    [S('list_of_doubles')]
);

```

Sample

```

select * from sample(
    ON ...
    SAMPLEFRACTION(f1[,f2,f3,...])
    [Seed('seed')]
select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    ApproximateSampleSize('size')
    [Seed('seed')]
select * from sample (
    ON ...
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1'[, 'cond2', ...])
    SAMPLEFRACTION(f1[,f2,f3,...])
    [SEED('seed')]
select * from sample (
    ON ...
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1'[, 'cond2', ...])
    SAMPLEFRACTION(f1[,f2,f3,...])
    [SEED('seed')]

```

```

select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1'[, 'cond2', ...])
    ApproximateSampleSize('total_sample_size')
    [SEED('seed')]
select * from sample (
    ON ... as data PARTITION BY ANY
    ON ... as summary DIMENSION
    CONDITIONONCOLUMN('column')
    CONDITIONON('cond1'[, 'cond2', ...])
    APPROXIMATESAMPLESIZE('s1'[, 's2', ...])
    [SEED('seed')]

```

FMeasure

```

select * from FMeasure
(
    ON {table_name|view_name|(query)}      PARTITION BY 1
    ExpectColumn('expect_column_name')
    PredictColumn('result_column_name')
    [Beta(beta_value)]
)
;
```

Text Analysis

Levenshtein Distance

```

SELECT *
FROM ldist
(
    ON {table_name | view_name | (query)}
    SOURCE (column1 [, column2,...])
    TARGET(column1)
    [THRESHOLD(value)]
    [OUTPUT_COLUMN_NAME(column_name)]
    [TARGET_COLUMN_NAME(column_name)]
    [SOURCE_COLUMN_NAME(column_name)]
    [ACCUMULATE(column1 [, column2,...])]
)
;
```

nGram

```

SELECT *
FROM nGram
(
    ON {table_name | view_name | (query)}
    TEXT_COLUMN('column_name')
    [DELIMITER('delimiter_regular_expression')]
    GRAMS(gram_number)
    [OVERLAPPING({'true'|'false'})]
    [CASE_INSENSITIVE({'true'|'false'})]
    [PUNCTUATION('punctuation_regular_expression')]
)
```

```

    RESET('reset_regular_expression')
[TOTAL]
[ACCUMULATE('column_name [, ...]')]
[NGRAM_COLUMN_NAME('column_name')]
[COUNT_COLUMN_NAME('column_name')]
) ;

```

Text Classifier

TextClassifierTrainer

```

SELECT *
FROM TextClassifierTrainer(
    ON (SELECT 1) PARTITION BY 1
    INPUTTABLE('inputTableName')
    TEXTCOLUMN('textColumnName')
    CATEGORYCOLUMN('catColumnName')
    MODEFILE('modelFileName')
    CLASSIFIERTYPE('classifierType')
    [CLASSIFIERPARAMETERS('name:value')]
    [NLPPARAMETERS('name:value'[, ...])]
    [FEATURESELECTION('DF:[min:max]')]
    [DATABASE('beehive')]
    [USERID('dbUserId')]
    PASSWORD('dbUserPassword')
    [DOMAIN('ip')])
) ;

```

TextClassifier

```

SELECT *
FROM TextClassifier(
    ON inputTable
    TEXTCOLUMN('columnName')
    Model('modelName')
    [ACCUMULATE('columnName1', 'columnName2', ..., 'columnNameN')]
) ;

```

TextClassifierEvaluator

```

SELECT * FROM TextClassifierEvaluator
(
    ON {table_name|view_name|(query)}
    EXPECTCOLUMN('expect_column_name')
    PREDICTCOLUMN('result_column_name')
    PARTITION BY 1
) ;

```

Text Parser (text_parser)

```

SELECT *
FROM text_parser
(
    ON {table_name|view_name|(query)}
    [PARTITION BY expression [, ...]]
    TEXT_COLUMN('text_column_name')
)

```

```

[CASE_INSENSITIVE('true' | 'false')]
[STEMMING('true' | 'false')]
[DELIMITER('delimiter_regular_expression')]
[TOTAL('true' | 'false')]
[PUNCTUATION('punctuation_regular_expression')]
[ACCUMULATE('column [, ...]')]
[TOKEN_COLUMN_NAME('token_column_name')]
[FREQUENCY_COLUMN_NAME('frequency_column_name')]
[TOTAL_COLUMN_NAME('total_column_name')]
[REMOVE_STOP_WORDS('true' | 'false')]
[POSITION_COLUMN_NAME('position_column_name')]
[LIST_POSITIONS('true' | 'false')]
[OUTPUT_BY_WORD('true' | 'false')]
[STEMMING_EXCEPTIONS('exception_rule_file')]
[STOP_WORDS('stop_word_file')]
);

```

Named Entity Recognition (NER)

FindNamedEntity

```

SELECT * FROM FindNamedEntity(
    ON {table_name|view_name|(query)} PARTITION BY ANY
    ON (configure_table) as ConfigureTable DIMENSION

    TEXT_COLUMN('text_column_name')
    MODEL('entity_type_name' [ ':' 'model_type' ':' 
        'model_file_name|regular_expression' [, ...]])
    [SHOW_CONTEXT('position_number')]
    [ENTITY_COLUMN ('column_name')]
    [OUTPUT_COLUMN ('column_name [, ...]))];

```

TrainNamedEntityFinder

```

SELECT * FROM TrainNamedEntityFinder
(
    ON {table_name|view_name|query}
    PARTITION BY 1
    TEXT_COLUMN('text_column_name')
    ENTITY_TYPE('entity_type')
    MODEL('model_name')
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    PASSWORD('password')
    [ITERATOR('iterator')]
    [CUTOFF('cutoff')]
)
;
```

EvaluateNamedEntityFinderRow and EvaluateNamedEntityFinderPartition

```

SELECT * FROM EvaluateNamedEntityFinderPartition(
    ON EvaluateNamedEntityFinderRow
    (
        ON {table_name|view_name|(query)}
        TEXT_COLUMN('text_column_name')

```

```

        MODEL ('model_data_file')
    )
PARTITION BY 1
) ;

```

Sentiment Extraction Functions

ExtractSentiment

```

SELECT *
FROM ExtractSentiment
(
    ON {table_name|view_name|(query)}
    TEXT_COLUMN('text_column_name')
    [MODEL('model_type[:model_file]')]
    [ONLY SUBJECTIVE_SENTENCE('{ true | false }')]
    [ACCUMULATE ('column [, ...]')]
    [LEVEL ('{DOCUMENT | SENTENCE}') ]
    [HIGH_PRIORITY('{ NEGATIVE_RECALL | NEGATIVE_PRECISION |
    POSITIVE_RECALL | POSITIVE_PRECISION | NONE }')]
    [FILTER('{POSITIVE | NEGATIVE| ALL}') ]
)
;
```

EvaluateSentimentExtractor

```

SELECT * FROM EvaluateSentimentExtractor(
    ON {table_name|view_name|(query)}
    EXPECT_COLUMN('expect_column_name')
    RESULT_COLUMN('result_column_name')
    PARTITION BY 1
)
;
```

TrainSentimentExtractor

```

SELECT *
FROM TrainSentimentExtractor
(
    ON {table_name|view_name|(query)}
    PARTITION BY 1
    [TEXT_COLUMN('text_column_name')]
    [SENTIMENT_COLUMN('category_column_name')]
    MODEL_FILE('model_name')
    [ONLY SUBJECTIVE_SENTENCE('{ true | false }')]
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    PASSWORD('password')
)
;
```

Naive Bayes Text Classifier

Naive Bayes Text (naiveBayesText)

```

CREATE TABLE model_table_name ( PARTITION KEY(token) ) AS
SELECT token, SUM( category_1 ) AS category_1, ... ,
SUM( category_n ) AS category_n FROM
naiveBayesText(

```

```

    ON input_table
    TEXT_COLUMN( text_column )
    CATEGORY_COLUMN( category_column )
    CATEGORIES( category_1, ... , category_n )
    [DELIMITER('delimiter_regular_expression')]
    [PUNCTUATION('punctuation_regular_expression'))]
GROUP BY token;

```

Naive Bayes Text Predict (naiveBayesTextPredict)

```

SELECT * FROM naiveBayesTextPredict(
    ON input_table
    MODEL_FILE( model_file_name )
    DOCUMENT_ID( document_id_column )
    TEXT_COLUMN( text_column )
    CATEGORIES( category_1, ... , category_n )
) ;

```

TextTokenizer

```

select * from TextTokenizer(
    on {table_name|view_name|query}
    TEXTCOLUMN('text_column_name')
    [LANGUAGE('language_type')]
    [OUTPUTDELIMITER('delimiter')]
    [OUTPUTBYWORD('true' | 'false')]
    [ACCUMULATE('accumulate_column_names')]
) ;

```

Cluster Analysis

kmeans

```

SELECT *
FROM kmeans
(
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('database_name')]
    [USERID('db_user')]
    [PASSWORD('password')]
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    NUMBERK(number_of_means)
    [MEANS(starting_clusters)]
    THRESHOLD(threshold)
    MAXITERNUM(max_iterations)
) ;

```

kmeansplot

```

SELECT *
FROM kmeansplot (
    ON inputtable PARTITION BY ANY

```

```

    ON centroids_table DIMENSION
    CENTROIDSTABLE('centroids_table')
);

```

Minhash

```

SELECT *
FROM minhash (
    ON (SELECT 1)
    PARTITION BY 1
    [DOMAIN('host_ip')]
    [DATABASE('db_name')]
    [USERID('user_id')]
    PASSWORD('password')
    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    COLUMNNAME('column_to_be_clustered')
    [SEEDTABLE('seed_table_to_use')]
    [SAVESEEDTO('seed_table_to_save')]
    NUMHASHFUNCTIONS('hash_function_number')
    KEYGROUPS('key_group_number')
    [HASH_INPUT_TYPE('bigint' | 'integer' | 'string' | 'hex')]
    [MINCLUSTERSIZE('minimum_cluster_size')]
    [MAXCLUSTERSIZE('maximum_cluster_size')]
);

```

Canopy

```

java -classpath canopydriver.jar:<classpath to file>
    -database=<database>
    -inputtable=<inputtable>
    -outputtable=<outputtable>
    -t1=<t1>
    -t2=<t2>
    -userid=<userid>
    -password=<password>
    -domain=<domain>

```

Naive Bayes

Naive Bayes Functions

naiveBayesMap and naiveBayesReduce

```

CREATE TABLE model_table_name (PARTITION KEY(column_name))
AS
SELECT * FROM naiveBayesReduce(
    ON (
        SELECT * FROM naiveBayesMap(
            ON input_table
            RESPONSE( 'response_column' )
            NUMERICINPUTS( 'numeric_input_columns' )
            CATEGORICALINPUTS( 'categorical_input_columns' )
        )
    )
)

```

```
)  
    PARTITION BY column_name  
) ;
```

naiveBayesPredict

```
SELECT *  
FROM naiveBayesPredict  
(  
    ON input_table  
    [ DOMAIN('queen_ip:port') ]  
    [ DATABASE('db_name') ]  
    [ USERID('user_id') ]  
    PASSWORD('db_password')  
    MODEL('model_table_name')  
    IDCOL('test_point_id_col')  
    NUMERICINPUTS('numeric_input_columns')  
    CATEGORICALINPUTS('categorical_input_columns')  
) ;
```

Decision Trees

Random Forest Functions

forest_drive

```
SELECT * FROM forest_drive(  
    ON (SELECT 1)  
    PARTITION BY 1  
  
    [DOMAIN('host:port')]  
    [DATABASE('database')]  
    [USERID('user_id')]  
    PASSWORD('password')  
  
    INPUTTABLE('input_table_name')  
    OUTPUTTABLE('output_table_name')  
  
    RESPONSE('response_column')  
    NUMERICINPUTS('numeric_input_columns')  
    CATEGORICALINPUTS('categorical_input_columns')  
  
    [TREETYPE(tree_type)]  
    [NUMTREES(number_of_trees)]  
    [TREESIZE(tree_size)]  
    [MINNODESIZE(min_node_size)]  
    [VARIANCE(variance)]  
    [MAXDEPTH(max_depth)]  
    [NUMSURROGATES(num_surrogates)]  
) ;
```

forest_predict

```
SELECT *
```

```

FROM forest_predict
(
    ON {table_name|view_name|(query)}
    [DOMAIN('host:port')]
    DATABASE('database_name')
    USERID('user_id')
    PASSWORD('password')
    MODELFIL('model_file')
    FOREST('model_table')
    NUMERICINPUTS('numeric_inputs')
    CATEGORICALINPUTS('categorical_inputs')
    IDCOL('id_column')
) ;

```

forest_analyze

```

SELECT * FROM forest_analyze(
    ON {table_name|view_name|(query)}
    [NUM_LEVELS(number_of_levels)]
) ;

```

Decision Trees

Single Decision Tree Functions

single_tree_drive

```

select * from single_tree_drive(
    ON (select 1) PARTITION BY 1
    [DOMAIN('host:port')]
    [DATABASE('database')]
    [USERID('user_id')]
    [PASSWORD('password')]
    ATTRIBUTETABLENAME('attribute_table_name')
    RESPONSETABLENAME('response_table_name')
    OUTPUTTABLENAME('output_table_name')
    [SPLITSTABLENAME('user_provided_splits_table_name')]
    [SPLITSVALUECOLUMN('splits_valcol1')]
    [NUMSPLITS('num_splits_to_consider')]
    [USEAPPROXIMATESPLITS(boolean)]
    [MATERIALIZESPLITSTABLEWITHNAME('table_name_for_splits')]
    [DROPTABLE(boolean)]
    [MINNODESIZE('minimum_split_size')]
    MAX_DEPTH('max_depth')
    [IMPURITYMEASUREMENT(Gini | Entropy | ChiSquare)]
    ATTRIBUTENAMECOLUMNS('att_col1' [, 'att_col2'] ...)
    ATTRIBUTEVALUECOLUMN('node_col')
    RESPONSECOLUMN('response_column_name')
    IDCOLUMNS('id_column1' [, idcolumn2])
);

```

single_tree_predict

```

SELECT * FROM single_tree_predict (
    ON attribute_table as attribute_table partition by pid_
    col1[, pid_col2[, ...]]
        ON model_table as model_table DIMENSION
        AttrTable_GroupbyColumns('gcol1'[, 'gcol2'...])
        AttrTable_pidColumns('pid_col1'[, 'pid_col2'...])
        AttrTable_valColumn('value_column')
        ModelTable_nodeColumn('node_column')
        ModelTable_sizeColumn('size_column')
        ModelTable_leftSizeColumn('left_size_column')
        ModelTable_rightSizeColumn('right_size_column')
        ModelTable_attrColumns('attr_1'[, 'attr_2'[, ...]])
        ModelTable_splitColumn('split_column')
        ModelTable_labelColumn('label_column')
        ModelTable_leftLabelColumn('left_label_column')
        ModelTable_rightLabelColumn('right_label_column'))
;

```

Association Analysis

Basket Generator (basket_generator)

```

SELECT *
FROM basket_generator
(
    ON {table_name | view_name | (query)}
    PARTITION BY expression [, ...]
    [BASKET_SIZE('basket_size_value')]
    BASKET_ITEM('basket_item_column')
    ACCUMULATE('column1 [, column2, ...]')
    COMBINATIONS('true'||'false')
    [ITEM_SET_MAX('item_set_max_value')]
);

```

Collaborative Filtering (cfILTER)

```

SELECT * FROM cfILTER (
    ON (SELECT 1)
    PARTITION BY 1
    [ domain('ip_address') ]
    [ database('db_name') ]
    [ userid('user_id') ]
    password('password')
    inputTable('input_table_name')
    outputTable('output_table_name')
    inputColumns('source_column1', 'source_column2', ...)
    joinColumns('join_column1', 'join_column2', ...)
    [ otherColumns('other_column1', 'other_column2', ...) ]
    [ partitionKeyColumn ('partitionKeyColumn1') ]
    [ maxSet('max_item_set') ]
    [ dropTable('true'||'false') ]
);

```

Graph Analysis

nTree

```
SELECT * FROM NTREE
(
    ON { input_table | view | query }
    PARTITION BY partition_columns
    [ORDER BY ordering_columns]
    ROOT_NODE(expression)
    NODE_ID(expression)
    PARENT_ID(expression)
    MODE('UP' | 'DOWN')
    ALLOW_CYCLES('true' | 'false')
    STARTS_WITH(expression | 'root' | 'leaf')
    OUTPUT('END' | 'ALL')
    RESULT(aggregate(expression) as alias)
    [LOGGING ('true' | 'false')]
)
;
```

Single Source Shortest Path (SSSP)

```
$ java
-classpath
path_1/ssspDriver.jar:
path_2/ncluster-sqlmr-api.jar:
path_3/noarch-ncluster-jdbc-driver.jar
com.asterdata.sqlmr.analytics.path_analysis.sssp.ssspDriver

-domain=ip_address
-database=database_name
-userid=user_id
-password=password

-inputTable=input_table_name
-sourceColumnName=source_column_name
-destinationColumnName=destination_column_name
-startNode=start_node_number

-outputTable=output_table_name
```

degrees (beta)

```
SELECT * FROM DEGREES (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
```

```
[ DIRECTED('true | t | false | f') ]
[ AUGMENTED('true | t | false | f') ]
);
```

triangle_finder (beta)

```
SELECT * FROM triangle_finder (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table_name')
    OUTPUTTABLE('output_table_name')
    [ STARTNODE('vertex1') ]
    [ ENDNODE('vertex2') ]
    [ OUTDEGREE('degree1') ]
    [ INDEGREE('degree2') ]
    [ INVOKEMETHOD('single | multiple') ]
)
;
```

rectangle_finder (beta)

```
SELECT * FROM rectangle_finder (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')
    [ STARTNODE('vertex1') ]
    [ ENDNODE('vertex2') ]
    [ OUTDEGREE('degree1') ]
    [ INDEGREE('degree2') ]
)
;
```

local_clustering_coefficient (beta)

```
SELECT * FROM local_clustering_coefficient (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    TRIANGLES('triangles_table')
)
;
```

```

    OUTPUTTABLE('output_table')
    [ NODECOL('node_column_name') ]
    [ DEGREECOL('degree_column_name') ]
);

```

pagerank (beta)

```

SELECT * FROM PAGERANK (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')
    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')

    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
    [ DAMPFACTOR('damping_factor') ]
    [ MAXITERNUM('max_iter') ]
    [ THRESHOLD('threshold') ]
    [ DELIMITER('delimiter') ]
);

```

eigen_centrality (beta)

```

SELECT * FROM EIGEN_CENTRALITY (
    ON (SELECT 1)
    PARTITION BY 1

    [ DOMAIN('host:port') ]
    [ DATABASE('database') ]
    [ USERID('user_id') ]
    PASSWORD('password')

    INPUTTABLE('input_table')
    OUTPUTTABLE('output_table')

    [ STARTNODE('source_column') ]
    [ ENDNODE('dest_column') ]
    [ MAXITERNUM('max_iter') ]
    [ THRESHOLD('threshold') ]
    [ DELIMITER('delimiter') ]
);

```

Data Transformation

Antiselect

```

SELECT *
FROM antiselect
(
    ON {table_name | view_name | (query)}
)

```

```
        EXCLUDE ('column_name' [, ...])
) ;
```

Multicase

```
SELECT *
  FROM multi_case
  (
  ON
  (
    SELECT *,
      Condition1 AS case1,
      Condition2 AS case2,
      ...,
      ConditionN AS caseN
    FROM {table_name|view_name|(query)}
  )
  LABELS
  (
    'case1 AS "label1"',
    'case2 AS "label2"',
    ...,
    'caseN AS "labelN"'
  )
) ;
```

Pack

```
SELECT *
  FROM pack
  (
  ON {table_name|view_name|(query)}
  [COLUMN_NAMES('column1' [, ...])]
  [COLUMN_DELIMITER('delimiter_value')]
  [INCLUDE_COLUMN_NAME('true'|'false')]
  PACKED_COLUMN_NAME('packed_column_name')
) ;
```

Unpack

```
SELECT *
  FROM unpack
  (
  ON {table_name | view_name | (query)}
  DATA_COLUMN('data_column')
  COLUMN_NAMES('column1' [, 'column2', ...])
  COLUMN_TYPES('datatype' [, 'datatype', ...])
  [COLUMN_DELIMITER('delimiter_value')]
  [DATA_PATTERN('data_pattern_regular_expression')]
  [DATA_GROUP('group_number')]
  [IGNORE_BAD_ROWS({'true'|'false'})]
) ;
```

Pivot

```
SELECT * FROM pivot(
```

```

    ON {table_name | view_name | (query)}
    PARTITION BY col1[, col2, ...]
    [ ORDER BY order_by_columns ]
    PARTITIONS('col1'[, 'col2', ...])
    ROWS('number_of_rows')
    PIVOT_KEYS('key1', 'key2'[, ...])
    PIVOT_COLUMN( 'pivot_column_name' )
    METRICS('metric_col1', 'metric_col2'[, ...])
);

```

Unpivot

```

SELECT * FROM unpivot
(
    ON input_timeseries_table
    [COLSTOUNPIVOT('column1', 'column2',...)] |
    [COLSTOUNPIVOTRANGE('[index1:index2]', '[index3:index4]', ...)]
    COLSTOACCUMULATE('column1', 'column2',...)
    [KEEPINPUTCOLUMNTYPES('true|false')]
    [ATTRIBUTECOLUMNNAME('column_name1')]
    [VALUECOLUMNNAME('column_name2')]
);

```

XMLParser

```

SELECT * FROM XMLParser(
    ON { table_name | view_name | (query) }
    TEXT_COLUMN('text_column_name')
    NODES('node_pair_string[,...]')
    [SIBLING('sibling_node_string[,...]')]
    [DELIMITER('delimiter_string')]
    [SIBLING_DELIMITER('delimiter_string')]
    [MAX_ITEM_NUMBER('max_item_number')]
    [ANCESTOR('nodes_path')]
    [OUTPUTCOLUMN_NODEID('column_name')]
    [OUTPUTCOLUMN_PARENT_NODE_NAME('column_name')]
    [ERROR_HANDLER('false|true[;[output_column_name:] column_names]')]
    [ACCUMULATE('column [, ...]')]
);

```

XMLRelation

```

SELECT *
FROM XMLRelation(
    TEXTCOLUMN ('text_column_name')
    DOCIDCOLUMNS ('docid_columns')
    [MAXDEPTH ('max_depth')]
    [EXCLUDEELEMENTS ('node_paths')]
    [ATTRIBUTETONODE ('false'|'true')]
    [ATTRIBUTEDELIMITER ('delimiter_string')]
    [OUTPUT ('fulldata'|'parentchild'|'fullpath')]
    [ERROR_HANDLER('false|true[;[output_column_name:] column_names]')]
    [ACCUMULATE ('column_names')]
);

```

```
) ;
```

JSONParser

```
SELECT * FROM JSONParser
(
    ON tablename
    TEXT_COLUMN ('text_columnname')
    NODES ('parentnode/childnode'[,...])
    [SEARCH_PATH('nodename/nodename/...')]
    [DELIMITER('delimiter_string')]
    [MAX_ITEM_NUMBER('number')]
    [NODEID_OUTPUTCOLUMN_NAME('columnname')]
    [PARENTNODE_OUTPUTCOLUMN_NAME('columnname')]
    [ACCUMULATE('columnname'[,...])]
    [ERROR_HANDLER('true|false'
        [[;output_col_name :] input_col_name1,
         input_col_name2, input_col_name3,...]])]
) ;
```

Apache Log Parser

```
SELECT *
FROM apache_log_parser
(ON {table_name|view_name|(query)}
 LOG_COLUMN('log_column_name')
 [LOG_FORMAT('format_string')]
 [EXCLUDE_FILES('.file_suffix[,.file_suffix2',...])
 [RETURN_SEARCH_INFO('true'|'false')])
) ;
```

outlierFilter

```
SELECT * FROM outlierFilter (
    ON (SELECT 1)
    PARTITION BY 1
    inputTable('input_table_name')
    outputTable('output_table_name')
    filterCol('data_col')
    [groupByCols('group_by_col1','group_by_col2',...)]
    [method({'percentile'}|'tukey'|'carling'|'MAD-median')]
    [useApproxPercentile('false'|'true')]
    [percentileValues('perc_lower', 'perc_upper')]
    [percentileTolerance ('accuracy')]
    [IQRmultiplier ('k')]
    [removeTail('upper'|'lower'|{'both'})]
    [replacementValue('newval')]
    [domain('ip_address')]
    [database('db_name')]
    [userid('user_id')]
    password('password')
) ;
```

IpGeo

```
IpGeo(
```

```

    ON {table_name|view_name|(query)}
    IPADDRESSCOLUMN('ip_address_column_name')
    CONVERTER('file_name','class_name')
    [IPDATABASELOCATION('location_of_the_geolocation_data-
base')]
    [ACCUMULATE('column_name_list')]
)

```

Visualization Functions

NpathViz

```

SELECT *
FROM NpathViz (
    ON <table|view|query> [AS 'input'] PARTITION BY i_col_
name
    [ON <table|view|query> [AS 'aggregates']
        ARTITION BY a_col_name]
    FREQUENCY_COL('score_col')
    PATH_COL('path_col')
    [GRAPH_TYPE('sankey'|'sigma'|'chord'|'tree'|'
        'gexf'|'graphviz')]
    [ACCUMULATE('col_name')]
    [ARGUMENTS('key1=value1','key2=value2','key3=value3')]
    [JUSTIFY('left'|'right')]
    [DIRECTED('true'|'false')]
    TITLE('graph_title')
    [SUBTITLE('graph_subtitle')]
    [SANKEY_SORTING('true')]
    [SANKEY_NODE_ORDER(node_list)]
    [SANKEY_NODE_ORDER_ARRAY('colNumber=>nodeList',...)]
);
    TITLE('Example 1')

```

CfilterViz

```

SELECT *
FROM CfilterViz (
    ON <table|view|query> [AS 'input'] PARTITION BY i_col_
name
    [ON <table|view|query> [AS 'aggregates']
        PARTITION BY a_col_name]
    SCORE_COL('score_col')
    ITEM1_COL('col1_item1')
    ITEM2_COL('col1_item2')
    CNT1_COL('cnt1')
    CNT2_COL('cnt2')
    [DIRECTED('true'|'false')]
    [GRAPH_TYPE('sigma'|'gexf'|'graphviz')]
    [ACCUMULATE('col_name')]
    [ARGUMENTS('key1=value1','key2=value2','key3=value3')]
    TITLE('Title')
    [SUBTITLE('Subtitle')]
);

```


Index

A

about this book 6
ACT
 install command 54
aggregate
 creating with nPath 157
 nPath aggregate 157
analytics
 nPath 156
Analytics Foundation
 download 52
anchor
 nPath 170
antiselect function 464
Apache log file configuration 506
Apache Log Parser
 web logs
 Apache Log Parser 505
Apache Log Parser Item-Name Mapping 506
Apache Log Parser Item-Name Mapping table 509
API
 installation 57
 nPath 156
application code
 install SQL-MapReduce application code 50
approx percentile 186
Approximate Distinct Count 183
Approximate Percentile 186
argument 29
 SQL-MapReduce function argument 29
association analysis 371
Aster Data, about 6
Aster Lens 546
Aster support portal 6
at-least operator 165
attribution analysis 85, 92
augmented edges table 441

B

Bayes' Theorem 380

C

canopy function 376
CART algorithm 386
CASE with multiple conditions 466

centroid 364
cfilter 418
CfilterViz 533
classification functions
 Naive Bayes 380
clickstream analysis
 nPath 156
 sequential pattern 104
clickstream example 173
cluster analysis 364
cluster analysis functions 364
clustering
 canopy 376
collaborative filtering function 418
column 469
 combine many columns into one 469
 split one column into many 471
 transpose columns into rows 474
combine many columns into one 469
comma operator in nPath 165
conventions 5
copyright 6
Correlation (stats correlation) 189
count_approx_distinct 183
Cumulative Moving Average 61
custom code
 nPath 156
customer support 6

D

data transformation
 JSON to relational table 499
data transformation functions 464
date of publication 6
decision trees 386
degrees function 441
degrees table 441
directed graph 441
DISTINCT
 in nPath 162
documentation conventions 5
documentation version and updates 6

E

edition 6
eigen_centrality function 458

eigenvector 223

F

file upload in Aster 50
filterViz function 533
finding patterns 156
forest_analyze 397
forest_drive 389
forest_predict 394
FrequentPaths function 104
function versioning 50
functions
 download 52
 installing driver-based 56
 nPath 156
 set permissions 53
 test 54
functions, detailed list of 43
functions, list of 40

G

Generalized Linear Model 205, 220
get latest documentation 6
GLM 205, 220
GRANT EXECUTE on function 53
graph analysis 436
 Single Source Shortest Path 436
graph functions
 degrees 441
 eigen_centrality 458
 local_clustering_coefficients 452
 pagerank 454
 rectangle_finder 448
 triangle_finder 444
GROUP BY
 in nPath 162
grouping
 canopy 376
 collaborative filtering 418

H

hash by locality 371
help 6
histogram function 191

I

install
 SQL-MapReduce function 50
install driver-based functions 56
install JDK client 57

J

Jaccard metric 371
JDK client
 installation 57
JSONParser function 499

K

k-Means 364
 kmeansplot supporting function 369
kmeansplot 369

L

LAG expression in nPath 166
Levenshtein distance 308
lexical tokenizer 310
LIMIT
 in nPath 162
Linear Regression 203
list all possible website paths 72
load
 install SQL-MapReduce application code 50
loading
 install SQL-MapReduce application code 50
local_clustering_coefficient function 452
locality-sensitive hashing 371

M

match mode in nPath 159
Minhash 371
multicase function 466
multi-gram 310

N

Naive Bayes classification function 380
Named Entity Recognition 328
nGram 310
n-gram 310
non-overlapping match 159
nonoverlapping match 159
nPath 156
 anchors 170
 describing patterns in 162
 examples 173
 introduction 156
 LAG expression 166
 MODE for overlap 159
 operators 169
 repeated patterns 165
 SYMBOLS clause 160
 syntax 158
NpathViz 522

nPathViz function 522

O

OFFSET

 in nPath 162

operators 169

 comma in nPath 165

 nPath 169

ORDER BY

 in nPath 162

overlapping match 159

P

Pack function 469

pagerank function 454

pairings, finding with collaborative filtering 418

PARTITION BY

 SQL-MapReduce and 26, 27

partitioning by canopy 376

path analysis

 list all possible website paths 72

path analysis functions 60

Path Generator 72

Path Generator function 72

Path Starter 75

Path Summarizer 78

pattern match, overlapping 159

pattern matching

 lag comparison 166

 nPath 156

 repeated pattern 165

 time series 166

patterns

 sequential pattern analysis function 104

PCA 223

pivot function 474

portal 6

predicate for an nPath symbol 160

previous row compared with current 166

principal component analysis 223

R

rectangle_finder function 448

relational analysis 414, 424

relational analysis functions 414, 424

repeated pattern matching 165

row 474

 transpose rows into columns 474

S

sample code

nPath 173

SAX 64

SAX function 64

SDK

 nPath 156

select all but listed columns 464

select: antiselect 464

Sessionization 82

similarity analysis 371

Simple Moving Average 227

Single Source Shortest Path 436

split into words 323

split one column into many 471

split text 310

SQL aggregate

 creating with nPath 157

SQL-MapReduce

 installing a function 50

 introduction 25

 syntax synopsis 27

SQL-MapReduce C SDK

 syntax synopsis 27

SQL-MR API

 installation 57

SQL-MR functions

 installing 52

 upgrading 53

 versioning 50

SSSP function 436

statistical analysis functions 182

stats glm 205, 220

stats linear reg 203

stats smavg 227

stats vwap 237

stats wmavg 230, 232

support 6

symbol predicate 160

 LAG expression 166

Symbolic Aggregate approXimation (SAX) 64

SYMBOLS clause 160

system utility functions 544

T

technical support 6

text

 split into words or grams 310

text analysis 310

 Named Entity Recognition 328

text analysis functions 308

text analytics

 Text Classifier 314

Text Classifier functions 314

text parser 323

text_parser 323
time series analysis
 Cumulative Moving Average 61
 SAX 64
time series functions 60
time series pattern matching 166
tokenize 310, 323
tokenize_cnt 323
traffic analysis 60
 attribution 85, 92, 156
 Path Generator 72
 Path Starter 75
 Path Summarizer 78
transformation functions 464
transpose columns into rows 474
triangle_finder function 444
TT TblTitle
 Table
 XMLParser Input 485, 486, 487
typeface conventions 5

U
undirected graph 441
unique words, finding 323
unpack function 471
updated documentation 6
upload
 install SQL-MapReduce application code 50
upload file to Aster 50
upload SQL-MapReduce function 50
URL 6
 Aster Support URL 6
user-defined function
 nPath 156
utilities
 nPath 156
utility functions 544

V
version
 documentation version 6
versioning of functions 50
Visualization Catalog Table 550
visualization functions
 CfilterViz 533
 filterViz 533
 NpathViz 522
 nPathViz 522
Visualization Tables 547
Volume-Weighted Average Price 237
vwap 237

W
website traffic analysis 60
 attribution 85, 92
 finding patterns 156
 Path Generator 72
 Path Starter 75
 Path Summarizer 78
Weighted Moving Average 230, 232
WHERE
 in nPath 162
wmavg 230, 232

X
XML
 data transformation 490
 flatten 490
 transform to relational table 481
XMLParser function 481
XMLRelation function 490