IBM® Netezza® Analytics
Release 3.3.5.0

Netezza Matrix Engine Reference Guide

Part Number 00J2222-03 Rev. 2

Note: Before using this information and the product that it supports, read the information in "Notices and Trademarks" on page 272.

Contents

Preface

Audience for This Guide....xxi

Purpose of This Guide....xxi

Conventions....xxi

If You Need Help....xxi

Comments on the Documentation....xxii

1 List of functions by category

analytic utilities....23
matrix operations....23

2 Reference Documentation: analytic utilities

APPLY_SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation Based on Previous Decomposition 29

3 Reference Documentation: matrix operations

ABS_ELEMENTS - Elementwise ABS 33
ABS_ELEMENTS - Elementwise ABS (entire matrix operation) 35
ADD - Matrix Addition 37
ALL_NONZERO - Test Whether all Matrix Element Values are Non-Zero 39
ANY_NONZERO - Test Whether any Matrix Element Values are Non-zero 40
BLOCK - Copy a Rectangular Block of a Matrix 42
CEIL_ELEMENTS - Elementwise Ceiling Function 44
CEIL_ELEMENTS - Elementwise Ceiling Function (entire matrix operation) 46
CHOOSE - Choose Operation 48
CONCAT - Concatenation 53
COPY_MATRIX - Copy a Matrix 56
COPY_SUBMATRIX - Copy a Rectangular Block of a Matrix 57
COVARIANCE - Matrix covariance calculation 59
CREATE_IDENTITY_MATRIX - Create an Identity Matrix 62
CREATE_MATRIX_FROM_TABLE - Create a Matrix from a row/column/value Table 63
CREATE_ONES_MATRIX - Create a Matrix of Ones 65
CREATE_RANDOM_CAUCHY_MATRIX - Create a random Matrix using Cauchy distributed random values 66

CREATE_RANDOM_EXPONENT_MATRIX - Create a random matrix using Exponential distributed random values 68
CREATE_RANDOM_GAMMA_MATRIX - Create a matrix of pseudorandom variables following the Gamma distribution 70
CREATE_RANDOM_LAPLACE_MATRIX - Create a matrix of pseudo-random variables following the Laplace distribution 72
CREATE_RANDOM_MATRIX - Matrix of Random, Uniformly Distributed Values 73
CREATE_RANDOM_NORMAL_MATRIX - Create a matrix of pseudorandom variables following the normal distribution 75
CREATE_RANDOM_POISSON_MATRIX - Create a matrix of pseudorandom variables following the Poisson distribution 77
CREATE_RANDOM_RAYLEIGH_MATRIX - Create a Matrix of random using a Rayleigh distributed random values generator 78
CREATE_RANDOM_UNIFORM_MATRIX - Create a matrix of pseudo-random variables following the uniform distribution 80
CREATE_RANDOM_WEIBULL_MATRIX - Create a matrix of pseudo-random variables following the Weibull distribution 82
CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix 84
CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix and export only non-empty cells 86
DEGREES_ELEMENTS - Elementwise Radians to Degrees Function 88
DEGREES_ELEMENTS - Elementwise Radians to Degrees Function (entire matrix operation) 90
DELETE_ALL_MATRICES - Deletes All Matrices 91
DELETE_MATRIX - Delete Matrix 92
DIAG - Diagonal 94
DIVIDE_ELEMENTS - Divide Matrices Element-by-element 96
EIGEN - Eigendecomposition 98
EQ - Elementwise Equal 100
EXP_ELEMENTS - Elementwise EXP Function 102
EXP_ELEMENTS - Elementwise EXP Function (entire matrix operation) 104
FLOOR_ELEMENTS - Elementwise Floor Function 106
FLOOR_ELEMENTS - Elementwise Floor Function (entire matrix operation) 108
GE - Elementwise Greater Than or Equal 110
GEMM - General Matrix Multiplication 112
GEMM - General Matrix Multiplication - simplified version 114
GET_NUM_COLS - Return the Number of Columns of a Matrix 116
GET_NUM_ROWS - Return the Number of Rows of a Matrix 117
GET_VALUE - Return the Value of a Matrix Element 118
GT - Elementwise Greater Than 119
INITIALIZE - Initializes nzMatrix 121
INSERT - Insert One Matrix into Another 122

INT_ELEMENTS - Elementwise Truncate Function 124
INT_ELEMENTS - Elementwise Truncate Function (entire matrix operation) 126
INVERSE - Matrix Inversion 128
IS_INITIALIZED - Is Initialized 132
KILL_ENGINE - Kill the Matrix Engine 132
KRONECKER - Kronecker Product 133
LE - Elementwise less than or equal 135
LINEAR_COMBINATION - Linear Combination of Matrix Components 137
LIST_MATRICES - Lists all Matrices in the Connected Database 143
LN_ELEMENTS - Elementwise LN Function 144
LN_ELEMENTS - Elementwise LN Function (entire matrix operation) 146
LOC - Locate Non-zero Elements 148
LOG_ELEMENTS - Elementwise log Function of any base 149
LOG_ELEMENTS - Elementwise log Function of any base for the specified block of elements ...151
LOG_ELEMENTS - Elementwise log Function of base 10 153
LT - Elementwise Less Than 155
MATRIX_EXISTS - Check if a Matrix Exists 157
MATRIX_VECTOR_OPERATION - Elementwise Matrix-vector Operation 158
MAX - Elementwise Maximum, Elementwise Logical OR 161
MIN - Elementwise Minimum, Elementwise Logical AND 163
MOD_ELEMENTS - Elementwise MOD Function 165
MOD_ELEMENTS - Elementwise MOD Function (entire matrix operation) 167
MTX_LINEAR_REGRESSION - Linear Regression 168
MTX_LINEAR_REGRESSION_APPLY - Linear Regression Model Applier 171
MTX_PCA - Principal Component Analysis (PCA) 173
MTX_PCA - Principal Component Analysis (PCA) - Non-storing Individual Observations Version 178
MTX_PCA - Principal Component Analysis (PCA) - Simplified Version 181
MTX_PCA_APPLY - PCA Model Applier 184
MTX_POW - nth Power of a Matrix 186
MTX_POW2 - nth Power of a Matrix 188
MULTIPLY_ELEMENTS - Multiply Matrices Element-by-element 189
NE - Elementwise Not Equal 191
NORMAL - Matrix of Random, Normally Distributed Values 194
NORMAL - Matrix of Random, Normally Distributed Values - Simplified Version 195
POWER_ELEMENTS - Elementwise POWER Function 197
POWER_ELEMENTS - Elementwise POWER Function (entire matrix operation) 199
PRINT - Print a Matrix 201
PRINT - Print a Matrix - Simplified Version 202
RADIANS_ELEMENTS - Elementwise RADIANS Function 203

RADIANS_ELEMENTS - Elementwise RADIANS Function (entire matrix operation) 206
RCV2SIMPLE - Transforms a row/column/value table to a "Simple" Matrix Table 208
RCV2SIMPLE_NUM - Transforms a row/column/value Table to a "Simple" Matrix Table 214
RED_MAX - Maximum Value of a Matrix 218
RED_MAX_ABS - Maximum Absolute Value of a Matrix 219
RED_MIN - Minimum Value of a Matrix 220
RED_MIN_ABS - Minimum Absolute Value of a Matrix 221
RED_SSQ - Sum of Squares of Values of a Matrix 222
RED_SUM - Sum Values of a Matrix 223
RED_TRACE - Trace 224
REDUCE_TO_VECT - Reduce to vector 225
REDUCTION - Reductions MAX MIN SSQ SUM TRACE 226
REMOVE - Remove Operation 227
REPEAT - Matrix Repeat 230
ROUND_ELEMENTS - Elementwise ROUND Function 232
ROUND_ELEMENTS - Elementwise ROUND Function (entire matrix operation) 234
SCALAR_OPERATION - Elementwise Scalar Operation 236
SCALAR_OPERATION - Elementwise Scalar Operation (entire matrix operation) 238
SCALE - Scale the Elements of a Matrix 240
SET_BLOCK_SIZE - Set the Block Size for the Data Distribution 242
SET_GRID_SIZE - Set the Process Grid Size for the Matrix Engine. 243
SET_GRID_SIZE_WITH_REDISTRIBUTE - Set the Process Grid Size for the Matrix Engine with Redistribution 244
SET_VALUE - Set the Value of a Matrix Element 245
SHAPE - Cyclically Fill a Matrix with Elements from a List 246
SHAPEMTX - Cyclically Fill a Matrix with Elements from a Row Vector 248
SIGN_REVERSE - Elementwise Sign Reversal 249
SIGN_REVERSE - Elementwise Sign Reversal (entire matrix operation) 252
SIMPLE2RCV - Transforms a "Simple" Matrix Table to row/column/value Representation 253
SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation 255
SOLVE - Solve the Matrix Equation A X = B 263
SOLVE_LINEAR_LEAST_SQUARES - Solve Linear Least Squares Problem 266
SQRT_ELEMENTS - Elementwise SQRT 268
SQRT_ELEMENTS - Elementwise SQRT (entire matrix operation) 270
SUBTRACT - Matrix Subtraction 271
SVD - Singular Value Decomposition 273
TRANSPOSE - Matrix Transpose 277
UNIFORM - Matrix of Random, Uniformly Distributed Values. 278
VEC_TO_DIAG - Create a Diagonal Matrix from a One-column Matrix 280
VECDIAG - Diagonal of a Matrix 281

Notices and Trademarks

Notices....284

Trademarks 285

Regulatory and Compliance 286

- Regulatory Notices....286
- Homologation Statement....286
- FCC - Industry Canada Statement....286
- CE Statement (Europe)....286
- VCCI Statement....286

Index

Preface

This guide describes the IBM Netezza Analytics Matrix Engine Package.

Audience for This Guide

This guide is written for developers who intend to use the IBM Netezza Analytics Matrix Engine Package with their IBM Netezza systems. It does not provide a tutorial on matrix concepts, linear algebra, or statistics, which you should be familiar with, depending on your needs. You should also have a basic understanding of the IBM Netezza system. For more information, see the *Netezza Matrix Engine Developer's Guide*.

Purpose of This Guide

This guide describes the stored procedures of the IBM Netezza Analytics Matrix Engine Package. The package provides matrix functions that can be used on the IBM Netezza database warehouse appliance.

Conventions

Note on Terminology: The terms User-Defined Analytic Process (UDAP) and Analytic Executable (AE) are synonymous.

The following conventions apply:

- *Italics* for emphasis on terms and user-defined values, such as user input.
- Upper case for SQL commands, for example, INSERT or DELETE.
- Bold for command line input, for example, **nzsystem stop**.
- Bold to denote parameter names, argument names, or other named references.
- Angle brackets (< >) to indicate a placeholder (variable) that should be replaced with actual text, for example, **nzmat <- nz.matrix("<matrix_name>")**.
- A single backslash ("\") at the end of a line of code to denote a line continuation. Omit the backslash when using the code at the command line, in a SQL command, or in a file.
- When referencing a sequence of menu and submenu selections, the ">" character denotes the different menu options, for example *Menu Name > Submenu Name > Selection*.

If You Need Help

If you are having trouble using the IBM Netezza appliance, IBM Netezza Analytics or any of its components:

1. Retry the action, carefully following the instructions in the documentation.
2. Go to the IBM Support Portal at http://www.ibm.com/support. Log in using your IBM ID and password. You can search the Support Portal for solutions. To submit a support request, click the 'Service Requests & PMRs' tab.
3. If you have an active service contract maintenance agreement with IBM, you can contact

customer support teams via telephone. For individual countries, please visit the Technical Support section of the IBM Directory of worldwide contacts: http://www14.software.ibm.com/webapp/set2/sas/f/handbook/contacts.html#phone.

Comments on the Documentation

We welcome any questions, comments, or suggestions that you have for the IBM Netezza documentation. Please send us an e-mail message at netezza-doc@wwpdl.vnet.ibm.com and include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

We appreciate your comments.

CHAPTER 1
List of functions by category

analytic utilities

APPLY_SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation Based on Previous Decomposition

matrix operations

ABS_ELEMENTS - Elementwise ABS

ABS_ELEMENTS - Elementwise ABS (entire matrix operation)

ADD - Matrix Addition

ALL_NONZERO - Test Whether all Matrix Element Values are Non-Zero

ANY_NONZERO - Test Whether any Matrix Element Values are Non-zero

BLOCK - Copy a Rectangular Block of a Matrix

CEIL_ELEMENTS - Elementwise Ceiling Function

CEIL_ELEMENTS - Elementwise Ceiling Function (entire matrix operation)

CHOOSE - Choose Operation

CONCAT - Concatenation

COPY_MATRIX - Copy a Matrix

COPY_SUBMATRIX - Copy a Rectangular Block of a Matrix

COVARIANCE - Matrix covariance calculation

CREATE_IDENTITY_MATRIX - Create an Identity Matrix

CREATE_MATRIX_FROM_TABLE - Create a Matrix from a row/column/value Table

CREATE_ONES_MATRIX - Create a Matrix of Ones

CREATE_RANDOM_CAUCHY_MATRIX - Create a random Matrix using Cauchy distributed random val-

ues

CREATE_RANDOM_EXPONENT_MATRIX - Create a random matrix using Exponential distributed random values

CREATE_RANDOM_GAMMA_MATRIX - Create a matrix of pseudorandom variables following the Gamma distribution

CREATE_RANDOM_LAPLACE_MATRIX - Create a matrix of pseudo-random variables following the Laplace distribution

CREATE_RANDOM_MATRIX - Matrix of Random, Uniformly Distributed Values

CREATE_RANDOM_NORMAL_MATRIX - Create a matrix of pseudorandom variables following the normal distribution

CREATE_RANDOM_POISSON_MATRIX - Create a matrix of pseudorandom variables following the Poisson distribution

CREATE_RANDOM_RAYLEIGH_MATRIX - Create a Matrix of random using a Rayleigh distributed random values generator

CREATE_RANDOM_UNIFORM_MATRIX - Create a matrix of pseudo-random variables following the uniform distribution

CREATE_RANDOM_WEIBULL_MATRIX - Create a matrix of pseudo-random variables following the Weibull distribution

CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix

CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix and export only non-empty cells

DEGREES_ELEMENTS - Elementwise Radians to Degrees Function

DEGREES_ELEMENTS - Elementwise Radians to Degrees Function (entire matrix operation)

DELETE_ALL_MATRICES - Deletes All Matrices

DELETE_MATRIX - Delete Matrix

DIAG - Diagonal

DIVIDE_ELEMENTS - Divide Matrices Element-by-element

EIGEN - Eigendecomposition

EQ - Elementwise Equal

EXP_ELEMENTS - Elementwise EXP Function

EXP_ELEMENTS - Elementwise EXP Function (entire matrix operation)

FLOOR_ELEMENTS - Elementwise Floor Function

FLOOR_ELEMENTS - Elementwise Floor Function (entire matrix operation)

GE - Elementwise Greater Than or Equal

GEMM - General Matrix Multiplication

GEMM - General Matrix Multiplication - simplified version

GET_NUM_COLS - Return the Number of Columns of a Matrix
GET_NUM_ROWS - Return the Number of Rows of a Matrix
GET_VALUE - Return the Value of a Matrix Element
GT - Elementwise Greater Than
INITIALIZE - Initializes nzMatrix
INSERT - Insert One Matrix into Another
INT_ELEMENTS - Elementwise Truncate Function
INT_ELEMENTS - Elementwise Truncate Function (entire matrix operation)
INVERSE - Matrix Inversion
IS_INITIALIZED - Is Initialized
KILL_ENGINE - Kill the Matrix Engine
KRONECKER - Kronecker Product
LE - Elementwise less than or equal
LINEAR_COMBINATION - Linear Combination of Matrix Components
LIST_MATRICES - Lists all Matrices in the Connected Database
LN_ELEMENTS - Elementwise LN Function
LN_ELEMENTS - Elementwise LN Function (entire matrix operation)
LOC - Locate Non-zero Elements
LOG_ELEMENTS - Elementwise log Function of any base
LOG_ELEMENTS - Elementwise log Function of any base for the specified block of elements
LOG_ELEMENTS - Elementwise log Function of base 10
LT - Elementwise Less Than
MATRIX_EXISTS - Check if a Matrix Exists
MATRIX_VECTOR_OPERATION - Elementwise Matrix-vector Operation
MAX - Elementwise Maximum, Elementwise Logical OR
MIN - Elementwise Minimum, Elementwise Logical AND
MOD_ELEMENTS - Elementwise MOD Function
MOD_ELEMENTS - Elementwise MOD Function (entire matrix operation)
MTX_LINEAR_REGRESSION - Linear Regression
MTX_LINEAR_REGRESSION_APPLY - Linear Regression Model Applier
MTX_PCA - Principal Component Analysis (PCA)
MTX_PCA - Principal Component Analysis (PCA) - Non-storing Individual Observations Version
MTX_PCA - Principal Component Analysis (PCA) - Simplified Version
MTX_PCA_APPLY - PCA Model Applier
MTX_POW - nth Power of a Matrix

MTX_POW2 - nth Power of a Matrix

MULTIPLY_ELEMENTS - Multiply Matrices Element-by-element

NE - Elementwise Not Equal

NORMAL - Matrix of Random, Normally Distributed Values

NORMAL - Matrix of Random, Normally Distributed Values - Simplified Version

POWER_ELEMENTS - Elementwise POWER Function

POWER_ELEMENTS - Elementwise POWER Function (entire matrix operation)

PRINT - Print a Matrix

PRINT - Print a Matrix - Simplified Version

RADIANS_ELEMENTS - Elementwise RADIANS Function

RADIANS_ELEMENTS - Elementwise RADIANS Function (entire matrix operation)

RCV2SIMPLE - Transforms a row/column/value table to a "Simple" Matrix Table

RCV2SIMPLE_NUM - Transforms a row/column/value Table to a "Simple" Matrix Table

RED_MAX - Maximum Value of a Matrix

RED_MAX_ABS - Maximum Absolute Value of a Matrix

RED_MIN - Minimum Value of a Matrix

RED_MIN_ABS - Minimum Absolute Value of a Matrix

RED_SSQ - Sum of Squares of Values of a Matrix

RED_SUM - Sum Values of a Matrix

RED_TRACE - Trace

REDUCE_TO_VECT - Reduce to vector

REDUCTION - Reductions MAX MIN SSQ SUM TRACE

REMOVE - Remove Operation

REPEAT - Matrix Repeat

ROUND_ELEMENTS - Elementwise ROUND Function

ROUND_ELEMENTS - Elementwise ROUND Function (entire matrix operation)

SCALAR_OPERATION - Elementwise Scalar Operation

SCALAR_OPERATION - Elementwise Scalar Operation (entire matrix operation)

SCALE - Scale the Elements of a Matrix

SET_BLOCK_SIZE - Set the Block Size for the Data Distribution

SET_GRID_SIZE - Set the Process Grid Size for the Matrix Engine.

SET_GRID_SIZE_WITH_REDISTRIBUTE - Set the Process Grid Size for the Matrix Engine with Redistribution

SET_VALUE - Set the Value of a Matrix Element

SHAPE - Cyclically Fill a Matrix with Elements from a List
SHAPEMTX - Cyclically Fill a Matrix with Elements from a Row Vector
SIGN_REVERSE - Elementwise Sign Reversal
SIGN_REVERSE - Elementwise Sign Reversal (entire matrix operation)
SIMPLE2RCV - Transforms a "Simple" Matrix Table to row/column/value Representation
SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation
SOLVE - Solve the Matrix Equation A X = B
SOLVE_LINEAR_LEAST_SQUARES - Solve Linear Least Squares Problem
SQRT_ELEMENTS - Elementwise SQRT
SQRT_ELEMENTS - Elementwise SQRT (entire matrix operation)
SUBTRACT - Matrix Subtraction
SVD - Singular Value Decomposition
TRANSPOSE - Matrix Transpose
UNIFORM - Matrix of Random, Uniformly Distributed Values.
VEC_TO_DIAG - Create a Diagonal Matrix from a One-column Matrix
VECDIAG - Diagonal of a Matrix

CHAPTER 2
Reference Documentation: analytic utilities

APPLY_SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation Based on Previous Decomposition

Transforms a table to row/column/value representation with nominal attributes decomposition based on previous decomposition. For each factor of a nominal attribute creates a separate column.

Usage

The APPLY_SIMPLE2RCV_ADV stored procedure has the following syntax:

- **APPLY_SIMPLE2RCV_ADV(NVARCHAR(ANY) paramString)**
  - Parameters
    - **paramString**
      The input parameters specification.

      Type: NVARCHAR(ANY)
    - **outtable**
      The name of the output RCV table.

      Type: NVARCHAR(ANY)
    - **inmeta**
      The name of the input meta table.

      Type: NVARCHAR(ANY)
    - **intable**
      The name of the input table.

      Type: NVARCHAR(ANY)
    - **id**
      The name of the column with unique values representing the ID.

Type: NVARCHAR(ANY)

- Returns
  INTEGER

Examples

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE);

INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);

INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);

INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);

-- Use columns V1, V2, and V3

CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1,
outmeta=RCV_META1, intable=SIMPLE1,
incolumnlist=V1;V2;V3, id=ID');

CALL NZM..APPLY_SIMPLE2RCV_ADV('outtable=RCV2,
inmeta=RCV_META1, intable=SIMPLE1, id=ID');

SELECT 'SIMPLE1',* FROM SIMPLE1 ORDER BY ID;

SELECT 'RCV1',* FROM RCV1 ORDER BY ROW, COL;

SELECT 'RCV_META1',* FROM RCV_META1 ORDER BY COLID;

SELECT 'RCV2',* FROM RCV2 ORDER BY ROW, COL;

DROP TABLE SIMPLE1;

DROP TABLE RCV1;

DROP TABLE RCV_META1;

DROP TABLE RCV2;
```

```
 SIMPLE2RCV_ADV

----------------

              3

(1 row)

 APPLY_SIMPLE2RCV_ADV

----------------------

                    3

(1 row)

 ?COLUMN? | ID |    V1    |    V2    |    V3
```

```
----------+----+--------+--------+--------
 SIMPLE1  |  1 | 100001 | 100002 | 100003
 SIMPLE1  |  4 | 200001 | 200002 | 200003
 SIMPLE1  |  9 | 300001 | 300002 | 300003
(3 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV1     |   1 |   1 | 100001
 RCV1     |   1 |   2 | 100002
 RCV1     |   1 |   3 | 100003
 RCV1     |   2 |   1 | 200001
 RCV1     |   2 |   2 | 200002
 RCV1     |   2 |   3 | 200003
 RCV1     |   3 |   1 | 300001
 RCV1     |   3 |   2 | 300002
 RCV1     |   3 |   3 | 300003
(9 rows)
 ?COLUMN?  | COLID | COLNAME | COLDICT | OUTCOLBEG | OUTCOLEND
-----------+-------+---------+---------+-----------+-----------
 RCV_META1 |     1 | V1      |         |         1 |         1
 RCV_META1 |     2 | V2      |         |         2 |         2
 RCV_META1 |     3 | V3      |         |         3 |         3
(3 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV2     |   1 |   1 | 100001
 RCV2     |   1 |   2 | 100002
 RCV2     |   1 |   3 | 100003
 RCV2     |   2 |   1 | 200001
 RCV2     |   2 |   2 | 200002
 RCV2     |   2 |   3 | 200003
 RCV2     |   3 |   1 | 300001
 RCV2     |   3 |   2 | 300002
```

```
RCV2      |     3 |     3 | 300003
(9 rows)
```

Related Functions

- category analytic utilities

CHAPTER 3

Reference Documentation: matrix operations

ABS_ELEMENTS - Elementwise ABS

This procedure implements the elementwise absolute value calculation for the specified block of elements.

Usage

The ABS_ELEMENTS stored procedure has the following syntax:

- **ABS_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use for the calculation.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use for the calculation.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use for the calculation.

      Type: INT4

- **col_stop**
  The last column of the input matrix to use for the calculation.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..SHAPE('-1.0', 4, 4, 'A');
CALL nzm..ABS_ELEMENTS('A', 'B', 2, 2, 3, 3);
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 ABS_ELEMENTS
--------------
 t
(1 row)
                                    PRINT
---------------------------------------------------------
--------------------
 -- matrix: A --
-1, -1, -1, -1
-1, -1, -1, -1
-1, -1, -1, -1
-1, -1, -1, -1
(1 row)
                                  PRINT
---------------------------------------------------------
----------------
```

```
 -- matrix: B --
-1, -1, -1, -1
-1, 1, 1, -1
-1, 1, 1, -1
-1, -1, -1, -1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ABS_ELEMENTS - Elementwise ABS (entire matrix operation)

This procedure implements the elementwise absolute value calculation.

Usage

The ABS_ELEMENTS stored procedure has the following syntax:

- **ABS_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..SHAPE('-1.0', 4, 4, 'A');
CALL nzm..ABS_ELEMENTS('A', 'B');
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 ABS_ELEMENTS
--------------
 t
(1 row)
                                    PRINT
----------------------------------------------------------
--------------------
 -- matrix: A --
-1, -1, -1, -1
-1, -1, -1, -1
-1, -1, -1, -1
-1, -1, -1, -1
(1 row)
                            PRINT
----------------------------------------------------------
----
 -- matrix: B --
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
```

```
1, 1, 1, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ADD - Matrix Addition

This procedure computes C = A + B, where A, B, and C are matrices.

Usage

The ADD stored procedure has the following syntax:

- **ADD(matrixA, matrixB, matrixC);**
  - Parameters
    - **matrixA**
      The name of matrix A.

      Type: NVARCHAR(ANY)
    - **matrixB**
      The name of matrix B.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..SHAPE('1.0,2.0,3.0,4.0', 4, 4, 'A');
```

```
CALL nzm..ADD('A', 'A', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('C' );
 SHAPE
-------
 t
(1 row)
 ADD
-----
 t
(1 row)
                            PRINT
-----------------------------------------------------------
----
 -- matrix: C --
2, 4, 6, 8
2, 4, 6, 8
2, 4, 6, 8
2, 4, 6, 8
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ALL_NONZERO - Test Whether all Matrix Element Values are Non-Zero

This stored procedure determines if all values in a matrix are non-zero.

Usage

The ALL_NONZERO stored procedure has the following syntax:

- **ALL_NONZERO(matrixName);**
  - Parameters
    - **matrixName**
      A string representing the name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE Returns 1 if all values are non-zero; 0 otherwise.

Details

Note that this operation checks for exact zeros, and fails to recognize "approximated zeros." Therefore, if the input matrix is a result of some approximation operations that should produce zero, but instead deliver an approximation of zero, then the procedure returns 1, as it does not recognize all values as non-zero.

Examples

```
CALL nzm..SHAPE('6,0,9, 4,6,0',2,3,'AA');
SELECT nzm..ALL_NONZERO('AA');
CALL nzm..DELETE_MATRIX('AA' );
```

```
 SHAPE
-------
 t
(1 row)
 ALL_NONZERO
-------------
           0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

```
CALL nzm..delete_matrix('AA');
CALL nzm..SHAPE('7',2,3,'BB');
SELECT nzm..ALL_NONZERO('BB');
CALL nzm..DELETE_MATRIX('BB' );
 SHAPE
-------
 t
(1 row)
 ALL_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ANY_NONZERO - Test Whether any Matrix Element Values are Non-zero

This stored procedure checks if any values in a matrix are non-zero.

Usage

The ANY_NONZERO stored procedure has the following syntax:

- **ANY_NONZERO(matrixName);**
  - Parameters
    - **matrixName**

      A string representing the name of the matrix.

      Type: NVARCHAR(ANY)

- Returns
  DOUBLE Returns 1 if any value is non-zero; 0 otherwise.

Details

Note that this operation checks for exact zeros, and fails to recognize "approximated zeros." Therefore, if the input matrix is a result of some approximation operations that should produce zero, but instead deliver an approximation of zero, then the procedure returns 1, as it recognizes some values as non-zero.

Examples

```
CALL nzm..SHAPE('6,0,9, 4,6,0',2,3,'AA');
SELECT nzm..ANY_NONZERO('AA');
CALL nzm..DELETE_MATRIX('AA' );
 SHAPE
-------
 t
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)

CALL nzm..SHAPE('0',2,3,'BB');
SELECT nzm..ANY_NONZERO('BB');
CALL nzm..DELETE_MATRIX('BB' );
  SHAPE
-------
 t
(1 row)
 ANY_NONZERO
-------------
           0
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

BLOCK - Copy a Rectangular Block of a Matrix

This procedure creates a matrix from the specified rectangular block of the input matrix.

Usage

The BLOCK stored procedure has the following syntax:

- **BLOCK(matrixIn, matrixOut, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'A');
CALL nzm..BLOCK('A', 'B', 2, 2, 3, 3);
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
  SHAPE
-------
 t
(1 row)
 BLOCK
-------
 t
(1 row)
                              PRINT
---------------------------------------------------------------
----
 -- matrix: A --
0, 1, 2, 3
4, 5, 6, 7
8, 9, 10, 11
12, 13, 14, 15
(1 row)
           PRINT
----------------------------
 -- matrix: B --
5, 6
9, 10
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CEIL_ELEMENTS - Elementwise Ceiling Function

This procedure implements the elementwise ceiling function.

Usage

The CEIL_ELEMENTS stored procedure has the following syntax:

- **CEIL_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

Type: INT4

- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Details

The ceiling function outputs the smallest integer that is not less than the argument.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,11.11,12
.12,13.13,14.14,15.15,16.16',4,4,'A');

CALL nzm..CEIL_ELEMENTS('A', 'B', 2, 2, 3, 3);

CALL nzm..PRINT('A');

CALL nzm..PRINT('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 CEIL_ELEMENTS
---------------
 t
(1 row)
                                                PRINT
---------------------------------------------------------------
---------------------------------------
 -- matrix: A --
0, 1.1, 2.2, 3.3
4.4, 5.5, 6.6, 7.7
8.8, 9.9, 10.1, 11.11
12.12, 13.13, 14.14, 15.15
```

```
(1 row)
                                                    PRINT
-----------------------------------------------------------------
-------------------------------------------
 -- matrix: B --
0, 1.1, 2.2, 3.3
4.4, 6, 7, 7.7
8.8, 10, 11, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CEIL_ELEMENTS - Elementwise Ceiling Function (entire matrix operation)

This procedure implements the elementwise ceiling function.

Usage

The CEIL_ELEMENTS stored procedure has the following syntax:

- **CEIL_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.
      Type: NVARCHAR(ANY)

- **matrixOut**
  The name of the output matrix.
  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

The ceiling function outputs the smallest integer that is not less than the argument.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,11.11,12
.12,13.13,14.14,15.15,16.16',4,4,'A');
CALL nzm..CEIL_ELEMENTS('A', 'B');
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 CEIL_ELEMENTS
---------------
 t
(1 row)
                                                   PRINT
-----------------------------------------------------------------
----------------------------------------
 -- matrix: A --
0, 1.1, 2.2, 3.3
4.4, 5.5, 6.6, 7.7
8.8, 9.9, 10.1, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
                                PRINT
```

```
-----------------------------------------------------------
-----------
 -- matrix: B --
0, 2, 3, 4
5, 6, 7, 8
9, 10, 11, 12
13, 14, 15, 16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CHOOSE - Choose Operation

Implements the choose operation, which chooses between elements of A or B.

Usage

The CHOOSE stored procedure has the following syntax:

- **CHOOSE(expression, matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **expression**
      An expression choosing between A and B matrix elements.

      Type: NVARCHAR(ANY)
    - **matrixAname**
      The name of input matrix A.

Type: NVARCHAR(ANY)

- **matrixBname**
  The name of input matrix B.

  Type: NVARCHAR(ANY)

- **matrixCname**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

The procedure implements the choose operation, which chooses between elements of A or B depending on the expression. Matrices A and B must be the same shape. The output matrix C is given the same shape. Each element of C is either the corresponding element of A or B depending on the value of the expression. In the expression the current element of A can be referred to as "a.value" and the current element of B referred to as "b.value" as shown in the example. The user is responsible for ensuring the expression is well-formed. Matrix C must not exist prior to the operation. Warning: Use "coalesce" in the user-supplied expression for sparse matrices to work.

Examples

```
CALL nzm..shape('1,2,3,4,5,6,7,8,9',3,3,'AA');
CALL nzm..shape('1,15,5,7',3,3,'BB');
CALL nzm..choose('a.value > b.value','AA','BB','CC');
CALL nzm..print('AA');
CALL nzm..print('BB');
CALL nzm..print('CC');
CALL nzm..delete_matrix('AA');
CALL nzm..delete_matrix('BB');
CALL nzm..delete_matrix('CC');
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
```

```
(1 row)
 CHOOSE
--------
 t
(1 row)
                  PRINT
------------------------------------------
 -- matrix: AA --
1, 2, 3
4, 5, 6
7, 8, 9
(1 row)
                   PRINT
--------------------------------------------
 -- matrix: BB --
1, 15, 5
7, 1, 15
5, 7, 1
(1 row)
                   PRINT
--------------------------------------------
 -- matrix: CC --
1, 15, 5
7, 5, 15
7, 8, 9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

CALL nzm..shape('1,2,3,4,5,6,7,8,9',3,3,'AA');

CALL nzm..shape('1,15,5,7',3,3,'BB');

CALL nzm..choose('(a.value * b.value) < (a.value + b.value)', 'AA', 'BB', 'CC');

CALL nzm..print('CC');

CALL nzm..delete_matrix('AA');

CALL nzm..delete_matrix('BB');

CALL nzm..delete_matrix('CC');

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 CHOOSE
--------
 t
(1 row)
                   PRINT
--------------------------------------------
 -- matrix: CC --
1, 15, 5
```

```
7, 5, 15
5, 7, 9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

```
CALL nzm..shape('1,2,3,4,5,6,7,8,9',3,3,'AA');
CALL nzm..shape('1,15,5,7',3,3,'BB');
CALL  nzm..choose('(a.value < b.value) and (a.value >
(b.value - 10))', 'AA', 'BB', 'CC');
CALL nzm..print('CC');
CALL nzm..delete_matrix('AA');
CALL nzm..delete_matrix('BB');
CALL nzm..delete_matrix('CC');
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
```

```
(1 row)
 CHOOSE
--------
 t
(1 row)
                   PRINT
-------------------------------------------
 -- matrix: CC --
1, 15, 3
4, 1, 6
5, 7, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CONCAT - Concatenation

Implements concatenation, either vertical or horizontal, of the two matrices passed in the parameters.

Usage

The CONCAT stored procedure has the following syntax:

- **CONCAT(NVARCHAR(ANY) matrixIn1, NVARCHAR(ANY) matrixIn2, NVARCHAR(ANY) matrixOut, NVARCHAR(ANY) concat_type);**
  - Parameters
    - **matrixIn1**
      The name of the first matrix to be concatenated.

      Type: NVARCHAR(ANY)
    - **matrixIn2**
      The name of the second matrix to be concatenated.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name to use for the resulting concatenated matrix.

      Type: NVARCHAR(ANY)
    - **concat_type**
      The concatenation type. Valid values are 'v' and 'h'.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

With vertical concatenation, the number of columns remains constant. With horizontal concatenation, the number of rows remains constant.

Examples

```
CALL nzm..shape('1',3,3,'A');
CALL nzm..shape('2',3,3,'B');
CALL nzm..CONCAT('A', 'B', 'C', 'v');
CALL nzm..print('C');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
CALL nzm..delete_matrix('C');
 SHAPE
-------
 t
(1 row)
 SHAPE
```

```
-------
 t
(1 row)
 CONCAT
--------
 t
(1 row)
                              PRINT
-----------------------------------------------------------------
--
 -- matrix: C --
1, 1, 1
1, 1, 1
1, 1, 1
2, 2, 2
2, 2, 2
2, 2, 2
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

COPY_MATRIX - Copy a Matrix

This procedure creates a copy of the specified matrix.

Usage

The COPY_MATRIX stored procedure has the following syntax:

- **COPY_MATRIX(matrixIn, matrixOut)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..shape('1',3,3,'A');
CALL nzm..COPY_MATRIX('A', 'B');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 COPY_MATRIX
-------------
 t
(1 row)
                          PRINT
```

```
----------------------------------------
 -- matrix: B --
1, 1, 1
1, 1, 1
1, 1, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

COPY_SUBMATRIX - Copy a Rectangular Block of a Matrix

This procedure creates a matrix from the specified rectangular block of the input matrix. This is an wrapper for the BLOCK stored procedure.

Usage

The COPY_SUBMATRIX stored procedure has the following syntax:

- **COPY_SUBMATRIX(matrixIn, matrixOut, row_start, row_stop, col_start, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4

- **row_stop**
  The last row of the input matrix to use.

  Type: INT4
- **col_start**
  The first column of the input matrix to use.

  Type: INT4
- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

▲ Returns
BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4
,'A');
CALL nzm..copy_submatrix('A', 'B', 2, 3, 1, 4);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 COPY_SUBMATRIX
----------------
 t
(1 row)
                              PRINT
-----------------------------------------------------------
----------
 -- matrix: A --
0, 1, 2, 3
```

```
4, 5, 6, 7
8, 9, 10, 11
12, 13, 14, 15
(1 row)
                    PRINT
-----------------------------------------
 -- matrix: B --
4, 5, 6, 7
8, 9, 10, 11
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

COVARIANCE - Matrix covariance calculation

This procedure calculates the column covariance estimator of the specified matrix.

Usage

The COVARIANCE stored procedure has the following syntax:

- **COVARIANCE(inputMatrix, outputMatrix);**
  - Parameters
    - **inputMatrix**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **outputMatrix**
      The name of the output matrix.

Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Details

The procedure calculates the column covariance estimator of the specified matrix, by centering each column and performing X^T X multiplication, divided by (n-1), where n is the number of rows of the provided matrix.

Examples

```
CALL nzm..create_ones_matrix('A', 5, 5);
CALL nzm..set_value('A', 1, 2, 2);
CALL nzm..set_value('A', 1, 3, 3);
CALL nzm..covariance('A', 'ACOVARIANCE');
CALL nzm..PRINT('A');
CALL nzm..PRINT('ACOVARIANCE');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('ACOVARIANCE');
```

```
 CREATE_ONES_MATRIX
--------------------
 t
(1 row)
 SET_VALUE
-----------
 t
(1 row)
 SET_VALUE
-----------
 t
(1 row)
 COVARIANCE
------------
 t
```

```
(1 row)
                                                  PRINT
-------------------------------------------------------------------------------------------------
 -- matrix: A --
1, 2, 3, 1, 1
1, 1, 1, 1, 1
1, 1, 1, 1, 1
1, 1, 1, 1, 1
1, 1, 1, 1, 1
(1 row)
                                                             PRINT
---------------------------------------------------------------------------------------------------------------
 -- matrix: ACOVARIANCE --
0, 0, 0, 0, 0
0, 0.2, 0.4, 0, 0
0, 0.4, 0.8, 0, 0
0, 0, 0, 0, 0
0, 0, 0, 0, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_IDENTITY_MATRIX - Create an Identity Matrix

The procedure creates an identity matrix of the size specified.

Usage

The CREATE_IDENTITY_MATRIX stored procedure has the following syntax:

- **CREATE_IDENTITY_MATRIX(matrixOut, size)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **size**
      The number of rows and columns in the matrix.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Details

An identity matrix is a square matrix with values of one (1) along the main diagonal and values of zero (0) elsewhere.

Examples

```
CALL nzm..CREATE_IDENTITY_MATRIX('A', 5);
CALL nzm..PRINT('A');
CALL nzm..DELETE_MATRIX('A' );
```

```
  CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
                                         PRINT
----------------------------------------------------------
------------------------------
 -- matrix: A --
1, 0, 0, 0, 0
0, 1, 0, 0, 0
0, 0, 1, 0, 0
```

```
0, 0, 0, 1, 0
0, 0, 0, 0, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_MATRIX_FROM_TABLE - Create a Matrix from a row/column/value Table

The procedure creates a matrix from a row/column/value table.

Usage

The CREATE_MATRIX_FROM_TABLE stored procedure has the following syntax:

- **CREATE_MATRIX_FROM_TABLE(source_table, mat_name, numRows, numCols);**
  - Parameters
    - **source_table**
      The name of the input table.

      Type: NVARCHAR(ANY)
    - **mat_name**
      The name of matrix to be created.

      Type: NVARCHAR(ANY)
    - **numRows**
      The number of matrix rows.

      Type: INT4
    - **numCols**
      The number of matrix columns.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Details

WARNING: THIS PROCEDURE SILENTLY CREATES AN INVALID MATRIX IF FED AN INVALID INPUT. UNLESS YOU

ARE CERTAIN YOUR INPUT IS VALID (NO DUPLICATE ROW, COLUMN ENTRIES, ETC.), FOLLOW THE INSTRUCTIONS BELOW FOR CALLING NZM.._TEST_DENSE_VALID(). Creates a matrix from a table having the following schema: (row INTEGER, col INTEGER, value DOUBLE PRECISION). The row indices should range from 1 to numRows, inclusive and the column indices should range from 1 to numCols, inclusive. Any (row, col) pairs outside these ranges are ignored. Each (row, col) pair may appear at most once. Null values are converted to zeros. If the number of values is greater than numRows * numCols, an exception is generated. In case of input data sparse form missing cells will be added and filled with zeros. You can use the procedure nzm.._test_dense_valid(mat_name) to verify that a created matrix has the proper number of values and that the (row, column) index pairs are unique.

Examples

```
create table mytable (row INT4, col INT4, value DOUBLE);
insert into mytable values (1, 1, 11);
insert into mytable values (1, 2, 12);
insert into mytable values (2, 1, 21);
insert into mytable values (2, 2, 22);
call nzm..create_matrix_from_table('mytable', 'A', 2, 2);
call nzm..print('A');
drop table mytable;
CALL nzm..delete_matrix('A' );
```

```
 CREATE_MATRIX_FROM_TABLE
--------------------------
 t
(1 row)
             PRINT
-------------------------------
 -- matrix: A --
11, 12
21, 22
(1 row)
 DELETE_MATRIX
---------------
 t
```

```
(1 row)
```

Related Functions

- category matrix operations

CREATE_ONES_MATRIX - Create a Matrix of Ones

This procedure creates a matrix with all elements equal to 1.0.

Usage

The CREATE_ONES_MATRIX stored procedure has the following syntax:

- **CREATE_ONES_MATRIX(mat_name, numRows, numCols);**
  - Parameters
    - **mat_name**
      The matrix name.

      Type: NVARCHAR(ANY)
    - **numRows**
      The number of matrix rows.

      Type: INT4
    - **numCols**
      The number of matrix columns.

      Type: INT4
  - Returns
    BOOLEAN TRUE always.

Examples

```
call nzm..create_ones_matrix('A', 3, 3);
call nzm..print('A');
call nzm..delete_matrix('A' );
```

```
  CREATE_ONES_MATRIX
--------------------
 t
(1 row)
                    PRINT
-----------------------------------------
```

```
 -- matrix: A --
1, 1, 1
1, 1, 1
1, 1, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_CAUCHY_MATRIX - Create a random Matrix using Cauchy distributed random values

This procedure creates a new matrix filled with Cauchy distributed random values using the parameters: Beta and shift. The formula is as follows: x = Beta tan (u) + shift. The u is a successive random number of a uniform distribution over the interval (-Pi/2, Pi/2).

Usage

The CREATE_RANDOM_CAUCHY_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_CAUCHY_MATRIX(matrixOut, numberOfRows, numberOfColums, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **shift**

The value to be used for shift.

Type: DOUBLE

- **beta**
  The value to be used for Beta.

  Type: DOUBLE

- Returns
  BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_CAUCHY_MATRIX ('A', 3,5, 1.0, 0.1);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
  CREATE_RANDOM_CAUCHY_MATRIX
------------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
            5
(1 row)
 GET_NUM_ROWS
--------------
            3
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
```

```
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_EXPONENT_MATRIX - Create a random matrix using Exponential distributed random values

This procedure creates a new matrix filled with Exponential distributed random values using the parameters: Beta and shift. The formula is as follows: x = -Beta ln(u) + shift. The u is a successive random number of a uniform distribution over the interval (0, 1).

Usage

The CREATE_RANDOM_EXPONENT_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_EXPONENT_MATRIX(matrixOut, numberOfRows, numberOfColums, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **shift**
      The value to be used for shift.

      Type: DOUBLE
    - **beta**
      The value to be used for Beta.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_EXPONENT_MATRIX('A', 5, 10, 1.0, 0.1);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
  CREATE_RANDOM_EXPONENT_MATRIX
-------------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_GAMMA_MATRIX - Create a matrix of pseudorandom variables following the Gamma distribution

This procedure creates a new matrix filled with pseudo-random variables following the Gamma distribution the specified parameters Alpha (shape), shift (offset) and Beta (scalefactor). The Generation technique depends on the values of the parameters and may involve pseudo-random variable transformation or the acceptance/rejection method.

Usage

The CREATE_RANDOM_GAMMA_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_GAMMA_MATRIX(matrixOut, numberOfRows, numberOfCols, alpha, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **alpha**
      The value used for Alpha.

      Type: DOUBLE
    - **shift**
      The value used for shift.

      Type: DOUBLE
    - **beta**
      The value used for Beta.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_GAMMA_MATRIX('A', 5, 10, 0.5, 1.0, 0.1);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
 CREATE_RANDOM_GAMMA_MATRIX
----------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_LAPLACE_MATRIX - Create a matrix of pseudo-random variables following the Laplace distribution

This procedure creates a new matrix filled with Laplace distributed pseudo-random variables using the parameters shift and Beta. The formula is as follows: x = -Beta*ln(u1) + shift, u2 <= 1/2 Beta*ln(u1) + shift, u2 > 1/2 Where u1, u2 is a pair of successive random numbers of a uniform distribution over the interval (0, 1).

Usage

The CREATE_RANDOM_LAPLACE_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_LAPLACE_MATRIX(matrixOut, numberOfRows, numberOfCols, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **shift**
      The value to be used for shift.

      Type: DOUBLE
    - **beta**
      The value to be used for Beta.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_LAPLACE_MATRIX('A', 5, 10, 1.0,
0.1);
```

```
CALL nzm..GET_NUM_COLS('A');
```

```
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
  CREATE_RANDOM_LAPLACE_MATRIX
------------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_MATRIX - Matrix of Random, Uniformly Distributed Values

This procedure creates a new matrix filled with uniformly distributed random values greater than or equal to zero and less than 1.

Usage

The CREATE_RANDOM_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_MATRIX(matrixOut, numberOfRows, numberOfColumns)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Details

This procedure uses drand48_r.

Examples

```
CALL nzm..CREATE_RANDOM_MATRIX('A', 5, 15);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
 CREATE_RANDOM_MATRIX
----------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           15
(1 row)
 GET_NUM_ROWS
--------------
```

```
              5
(1 row)
 ANY_NONZERO
-------------
            1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_NORMAL_MATRIX - Create a matrix of pseudorandom variables following the normal distribution

This procedure creates a matrix of pseudorandom variables following the normal distribution using the parameters shift and sigma. The generation technique is based on the CDF inversion according to the following equation: x = sigma * SQRT(2) * Erf^{-1} (u) + shift.

Usage

The CREATE_RANDOM_NORMAL_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_NORMAL_MATRIX(matrixOut, numberOfRows, numberOfColumns, shift, sigma)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **shift**
      The value to be used for shift.

Type: DOUBLE

- **sigma**
  The value to be used for sigma.

  Type: DOUBLE

- Returns
  BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_NORMAL_MATRIX('A', 5, 10, 1.0,
0.1);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
  CREATE_RANDOM_NORMAL_MATRIX
-----------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
```

```
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_POISSON_MATRIX - Create a matrix of pseudorandom variables following the Poisson distribution

This procedure creates a new matrix filled with pseudorandom variables following Poisson distribution using the parameters Lambda and mean 1/Lambda.

Usage

The CREATE_RANDOM_POISSON_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_POISSON_MATRIX(matrixOut, numberOfRows, numberOfColumns, lambda)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **lambda**
      The value to be used for lambda.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_POISSON_MATRIX('A', 5, 10, 1.2345);
```

```
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );
```

```
 CREATE_RANDOM_POISSON_MATRIX
------------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_RAYLEIGH_MATRIX - Create a Matrix of random using a Rayleigh distributed random values generator

This procedure creates a new matrix filled with Rayleigh distributed random values using the parameters Beta and shift. The formula is as follows: x = Beta * SQRT(-ln(u)) + shift. The u is a successive random number of a uniform distribution over the interval (0, 1).

Usage

The CREATE_RANDOM_RAYLEIGH_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_RAYLEIGH_MATRIX(matrixOut, numberOfRows, numberOfCols, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4
    - **numberOfColumns**
      The number of matrix columns.

      Type: INT4
    - **shift**
      The value to be used for shift.

      Type: DOUBLE
    - **beta**
      The value to be used for Beta.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_RAYLEIGH_MATRIX('A', 5, 10, 1.0, 0.1);
CALL nzm..GET_NUM_COLS('A');
CALL nzm..GET_NUM_ROWS('A');
CALL nzm..ANY_NONZERO('A');
CALL nzm..DELETE_MATRIX ('A' );

 CREATE_RANDOM_RAYLEIGH_MATRIX
-------------------------------
```

```
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_UNIFORM_MATRIX - Create a matrix of pseudo-random variables following the uniform distribution

This procedure creates a new matrix of pseudo-random variables following the uniform distribution over the real interval [a,b].

Usage

The CREATE_RANDOM_UNIFORM_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_UNIFORM_MATRIX(matrixOut, numberOfRows, numberOfColumns, minVal, maxVal)**
  - Parameters

- **matrixOut**
  The name of the matrix to be generated.

  Type: NVARCHAR(ANY)
- **numberOfRows**
  The number of matrix rows.

  Type: INT4
- **numberOfColumns**
  The number of matrix columns.

  Type: INT4
- **minVal**
  The minimum value.

  Type: DOUBLE
- **maxValue**
  The maximum value.

  Type: DOUBLE

- Returns
  BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_UNIFORM_MATRIX('A', 5, 10, -9.98765,
9.98765);

CALL nzm..GET_NUM_COLS('A');

CALL nzm..GET_NUM_ROWS('A');

CALL nzm..ANY_NONZERO('A');

CALL nzm..DELETE_MATRIX ('A' );
```

```
 CREATE_RANDOM_UNIFORM_MATRIX
------------------------------
 t
(1 row)
 GET_NUM_COLS
--------------
           10
(1 row)
```

```
 GET_NUM_ROWS
--------------
            5
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_RANDOM_WEIBULL_MATRIX - Create a matrix of pseudo-random variables following the Weibull distribution

This procedure creates a new matrix filled with pseudo-random variables following Weibull distribution using the specified parameters Alpha, Beta and shift. The Generation technique is based on the CDF inversion according to following equation: x = Beta * POWER((-ln(u)), (1/Alfa)) + shift where u is a pseudo-random variable uniformly distributed over the interval (0, 1).

Usage

The CREATE_RANDOM_WEIBULL_MATRIX stored procedure has the following syntax:

- **CREATE_RANDOM_WEIBULL_MATRIX(matrixOut, numberOfRows, numberOfCols, alpha, shift, beta)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of matrix rows.

      Type: INT4

- **numberOfColumns**
  The number of matrix columns.

  Type: INT4
- **alpha**
  The value to be used as Alpha.

  Type: DOUBLE
- **shift**
  The value to be used as shift.

  Type: DOUBLE
- **beta**
  The value to be used as Beta.

  Type: DOUBLE

- Returns
  BOOLEAN TRUE if successful.

Details

This procedure uses the MKL library.

Examples

```
CALL nzm..CREATE_RANDOM_WEIBULL_MATRIX('A', 5, 10, 0.5, 1.0,
0.1);

CALL nzm..GET_NUM_COLS('A');

CALL nzm..GET_NUM_ROWS('A');

CALL nzm..ANY_NONZERO('A');

CALL nzm..DELETE_MATRIX ('A' );
```

```
 CREATE_RANDOM_WEIBULL_MATRIX
------------------------------
 t
(1 row)

 GET_NUM_COLS
--------------
           10
(1 row)

 GET_NUM_ROWS
--------------
            5
```

```
(1 row)
 ANY_NONZERO
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix

This procedure creates a user-visible table from a matrix.

Usage

The CREATE_TABLE_FROM_MATRIX stored procedure has the following syntax:

- **CREATE_TABLE_FROM_MATRIX(mat_name, destination_table);**
  - Parameters
    - **mat_name**
      The name of the matrix to be copied.

      Type: NVARCHAR(ANY)
    - **destination_table**
      The name of the table to be generated.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

This procedure creates a table, owned by the caller, having the following schema: (row INTEGER, col INTEGER, value DOUBLE PRECISION). The created table contains the matrix data in a row/column/value representation. This hides the implementation details of NZMatrix and provides data to the user via a table representation. This procedure exports all cells of the matrix.

Examples

```
CALL nzm..shape('1,2,3,4,5,6,7,8,9',3,3,'A');
call nzm..create_table_from_matrix('A', 'B');
select * from B order by row,col;
call nzm..delete_matrix('A' );
drop table B;
```

```
 SHAPE
-------
 t
(1 row)
 CREATE_TABLE_FROM_MATRIX
--------------------------
 t
(1 row)
 ROW | COL | VALUE
-----+-----+-------
   1 |   1 |     1
   1 |   2 |     2
   1 |   3 |     3
   2 |   1 |     4
   2 |   2 |     5
   2 |   3 |     6
   3 |   1 |     7
   3 |   2 |     8
   3 |   3 |     9
(9 rows)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

CREATE_TABLE_FROM_MATRIX - Create a User-visible Table from a Matrix and export only non-empty cells

This procedure creates a user-visible table from a matrix and allows it to export only non-empty cells, that is, cell with non-zero values.

Usage

The CREATE_TABLE_FROM_MATRIX stored procedure has the following syntax:

- **CREATE_TABLE_FROM_MATRIX(mat_name, destination_table, sparse_only);**
  - Parameters
    - **mat_name**
      The name of the matrix to be copied.

      Type: NVARCHAR(ANY)
    - **destination_table**
      The name of the table to be generated.

      Type: NVARCHAR(ANY)
    - **sparse_only**
      If TRUE, only non empty (non zero) values are exported.

      Type: BOOLEAN
  - Returns
    BOOLEAN TRUE, if successful.

Details

Creates a table, owned by the caller, having the following schema: (row INTEGER, col INTEGER, value DOUBLE PRECISION). The created table contains the matrix data in a row/column/value representation. This hides the implementation details of NZMatrix and provides data to the user via a table representation. This procedure can export only nonempty cells, that is, cells with a non-zero value.

Examples

```
call nzm..shape('0,1,2,0,0,0,0,0,33',3,3,'A');

call nzm..create_table_from_matrix('A',
'my_rcv_dense',false);

call nzm..create_table_from_matrix('A',
'my_rcv_sparse',true);
```

```
select * from my_rcv_dense order by row,col;
select * from my_rcv_sparse order by row,col;
drop table  my_rcv_dense;
drop table  my_rcv_sparse ;
call nzm..delete_matrix('A' );
```

```
  SHAPE
-------
 t
(1 row)
 CREATE_TABLE_FROM_MATRIX
--------------------------
 t
(1 row)
 CREATE_TABLE_FROM_MATRIX
--------------------------
 t
(1 row)
 ROW | COL | VALUE
-----+-----+-------
   1 |   1 |     0
   1 |   2 |     1
   1 |   3 |     2
   2 |   1 |     0
   2 |   2 |     0
   2 |   3 |     0
   3 |   1 |     0
   3 |   2 |     0
   3 |   3 |    33
(9 rows)
 ROW | COL | VALUE
-----+-----+-------
   1 |   2 |     1
```

```
   1 |    3 |      2
   3 |    3 |     33
(3 rows)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

DEGREES_ELEMENTS - Elementwise Radians to Degrees Function

This procedure implements an elementwise radians to degrees conversion.

Usage

The DEGREES_ELEMENTS stored procedure has the following syntax:

- **DEGREES_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**

The last column of the input matrix to use.

Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Details

The last four arguments may be omitted, in which case the procedure applies to the entire input matrix.

Examples

```
CALL nzm..SHAPE('0, 0.78539816339745, 1.5707963267949,
3.1415926535898, 4.7123889803847, 6.2831853071796',2,3,'A');

CALL nzm..DEGREES_ELEMENTS('A', 'B',2,1,2,3);

CALL nzm..PRINT ('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
  SHAPE
-------
 t
(1 row)
 DEGREES_ELEMENTS
------------------
 t
(1 row)
                                PRINT
---------------------------------------------------------------
-----
 -- matrix: B --
0, 0.78539816339745, 1.5707963267949
180, 270, 360
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

DEGREES_ELEMENTS - Elementwise Radians to Degrees Function (entire matrix operation)

This procedure implements an elementwise radians to degrees conversion.

Usage

The DEGREES_ELEMENTS stored procedure has the following syntax:

- **DEGREES_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..SHAPE('0, 0.78539816339745, 1.5707963267949,
3.1415926535898, 4.7123889803847,
6.2831853071796',2,3,'A');

CALL nzm..DEGREES_ELEMENTS('A', 'B');

CALL nzm..PRINT ('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );

 SHAPE
```

```
-------
 t
(1 row)
 DEGREES_ELEMENTS
------------------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: B --
0, 45, 90
180, 270, 360
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

DELETE_ALL_MATRICES - Deletes All Matrices

The procedure deletes all matrices in the current database.

Usage

The DELETE_ALL_MATRICES stored procedure has the following syntax:

- **DELETE_ALL_MATRICES();**
  - Returns

BOOLEAN TRUE always.

Examples

```
CALL nzm..SHAPE('0,1,2,3,4,5,6,7,8,9',3,3,'A');
CALL nzm..DELETE_ALL_MATRICES();
CALL nzm..LIST_MATRICES();
```

```
 SHAPE
-------
 t
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
 LIST_MATRICES
---------------

(1 row)
```

Related Functions

- category matrix operations

DELETE_MATRIX - Delete Matrix

Deletes a matrix.

Usage

The DELETE_MATRIX stored procedure has the following syntax:

- **DELETE_MATRIX(mat_name);**
  - Parameters
    - **mat_name**
      The name of the matrix to be deleted.

      Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Details

This procedure throws an exception if the specified matrix does not exist.

Examples

```
CALL nzm..SHAPE('0,1,2,3,4,5,6,7,8,9',3,3,'A');
CALL nzm..SHAPE('0,1,2,3,4,5,6,7,8,9',3,3,'B');
CALL nzm..DELETE_MATRIX('A');
CALL nzm..LIST_MATRICES();
CALL nzm..DELETE_MATRIX('B');
CALL nzm..LIST_MATRICES();
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 LIST_MATRICES
---------------
 B
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 LIST_MATRICES
```

```
---------------
(1 row)
```

Related Functions

- category matrix operations

DIAG - Diagonal

This procedure creates a diagonal matrix from the diagonal elements of the input matrix.

Usage

The DIAG stored procedure has the following syntax:

- **DIAG(NVARCHAR(ANY) matrixIn, NVARCHAR(ANY) matrixOut);**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4
,'A');
CALL nzm..DIAG('A', 'B');
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
 SHAPE
-------
```

```
 t
(1 row)
 DIAG
------
 t
(1 row)
                              PRINT
-----------------------------------------------------------------
----
 -- matrix: A --
0, 1, 2, 3
4, 5, 6, 7
8, 9, 10, 11
12, 13, 14, 15
(1 row)
                             PRINT
-----------------------------------------------------------------
 -- matrix: B --
0, 0, 0, 0
0, 5, 0, 0
0, 0, 10, 0
0, 0, 0, 15
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

DIVIDE_ELEMENTS - Divide Matrices Element-by-element

The procedure computes matrix C using element-by-element division of matrix A by matrix B: Cij = Aij / Bij.

Usage

The DIVIDE_ELEMENTS stored procedure has the following syntax:

- **DIVIDE_ELEMENTS(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixB**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..SHAPE('1,4,9,16,25,36,49,64,81',3,3,'A');
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9',3,3,'B');
CALL nzm..DIVIDE_ELEMENTS('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
```

```
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 DIVIDE_ELEMENTS
-----------------
 t
(1 row)
                        PRINT
-----------------------------------------
 -- matrix: C --
1, 2, 3
4, 5, 6
7, 8, 9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

EIGEN - Eigendecomposition

This procedure computes the eigenvalues and eigenvectors of a symmetric matrix.

Usage

The EIGEN stored procedure has the following syntax:

- **EIGEN(matrixA, matrixW, matrixZ);**
  - Parameters
    - **matrixA**
      The name of the matrix to be decomposed, referred to as matrix A.

      Type: NVARCHAR(ANY)
    - **matrixW**
      The name of the matrix to hold the eigenvalues, referred to as matrix W.

      Type: NVARCHAR(ANY)
    - **matrixZ**
      The name of the matrix to hold the eigenvalues, referred to as matrix Z.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..create_random_matrix('A0', 500, 500);
CALL nzm..create_ones_matrix('A1', 500, 500);
CALL nzm..add('A0', 'A1', 'A2');
CALL nzm..transpose('A2','A3');
CALL nzm..add('A2','A3','A');
CALL nzm..eigen('A', 'W', 'Z');
CALL nzm..delete_matrix('A0' );
CALL nzm..delete_matrix('A1' );
CALL nzm..delete_matrix('A2' );
CALL nzm..delete_matrix('A3' );
CALL nzm..delete_matrix('A' );
CALL nzm..delete_matrix('W' );
CALL nzm..delete_matrix('Z' );
```

```
 CREATE_RANDOM_MATRIX
----------------------
 t
(1 row)
 CREATE_ONES_MATRIX
--------------------
 t
(1 row)
 ADD
-----
 t
(1 row)
 TRANSPOSE
-----------
 t
(1 row)
 ADD
-----
 t
(1 row)
 EIGEN
-------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

EQ - Elementwise Equal

This procedure implements an elementwise computation of the C := A == B comparison, where A, B, and C are matrices.

Usage

The EQ stored procedure has the following syntax:

- **EQ(matrixAname,matrixBname,matrixCname);**
  - Parameters
    - **matrixAname**

The name of input matrix A.

Type: NVARCHAR(ANY)

- **matrixBname**
  The name of input matrix B.

  Type: NVARCHAR(ANY)

- **matrixCname**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same number of dimensions, that is, the same number of rows and columns. The output matrix C is given the same shape. Matrix C must not exist prior to the operation. Matrix C contains only zeros and ones, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..SHAPE('1,15,5,7', 3, 3, 'B');
CALL nzm..EQ('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 EQ
----
 t
```

```
(1 row)
                    PRINT
-----------------------------------------
 -- matrix: C --
1, 0, 0
0, 0, 0
0, 1, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

EXP_ELEMENTS - Elementwise EXP Function

This procedure implements the elementwise exponential value calculation for the specified block of elements.

Usage

The EXP_ELEMENTS stored procedure has the following syntax:

- **EXP_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters

- **matrixIn**
  The name of the input matrix.

  Type: NVARCHAR(ANY)
- **matrixOut**
  The name of the output matrix.

  Type: NVARCHAR(ANY)
- **row_start**
  The first row of the input matrix to use.

  Type: INT4
- **col_start**
  The first column of the input matrix to use.

  Type: INT4
- **row_stop**
  The last row of the input matrix to use.

  Type: INT4
- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

▲ Returns
BOOLEAN TRUE, if successful.

Details

The last four arguments may be omitted, in which case the procedure applies to the entire input matrix.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..EXP_ELEMENTS('A', 'B', 2, 2, 2, 2);
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 EXP_ELEMENTS
--------------
```

```
 t
(1 row)
                          PRINT
---------------------------------------------------------
 -- matrix: B --
1, 2, 3
4, 148.41315910258, 0
6, 7, 8
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

EXP_ELEMENTS - Elementwise EXP Function (entire matrix operation)

This procedure implements the elementwise exponential value calculation for the specified block of elements.

Usage

The EXP_ELEMENTS stored procedure has the following syntax:

- **EXP_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)

- **matrixOut**
  The name of the output matrix.
  Type: NVARCHAR(ANY)

▲ Returns
BOOLEAN TRUE, if successful.

Details

This procedure applies to the entire input matrix.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..EXP_ELEMENTS('A', 'B');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 EXP_ELEMENTS
--------------
 t
(1 row)

PRINT
----------------------------------------------------------------
----------------------------------------------------------------
--------------------------
 -- matrix: B --
2.718281828459, 7.3890560989307, 20.085536923188
54.598150033144, 148.41315910258, 1
403.42879349274, 1096.6331584285, 2980.9579870417
(1 row)
 DELETE_MATRIX
---------------
```

```
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

FLOOR_ELEMENTS - Elementwise Floor Function

This procedure implements an elementwise rounding to the next smallest integer.

Usage

The FLOOR_ELEMENTS stored procedure has the following syntax:

- **FLOOR_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,11.11,12
.12,13.13,14.14,15.15,16.16',4,4,'A');
CALL nzm..FLOOR_ELEMENTS('A', 'B', 2, 2, 3, 3);
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 FLOOR_ELEMENTS
----------------
 t
(1 row)
                                                PRINT
---------------------------------------------------------------
---------------------------------------
 -- matrix: A --
0, 1.1, 2.2, 3.3
4.4, 5.5, 6.6, 7.7
8.8, 9.9, 10.1, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
                                            PRINT
---------------------------------------------------------------
-------------------------------
 -- matrix: B --
```

```
0, 1.1, 2.2, 3.3
4.4, 5, 6, 7.7
8.8, 9, 10, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

FLOOR_ELEMENTS - Elementwise Floor Function (entire matrix operation)

This procedure implements elementwise rounding to the next smallest integer.

Usage

The FLOOR_ELEMENTS stored procedure has the following syntax:

- **FLOOR_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,11.11,12
.12,13.13,14.14,15.15,16.16',4,4,'A');
CALL nzm..FLOOR_ELEMENTS('A', 'B');
CALL nzm..PRINT('A');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 FLOOR_ELEMENTS
----------------
 t
(1 row)
                                                PRINT
---------------------------------------------------------------
---------------------------------------
 -- matrix: A --
0, 1.1, 2.2, 3.3
4.4, 5.5, 6.6, 7.7
8.8, 9.9, 10.1, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
                               PRINT
---------------------------------------------------------------
----
 -- matrix: B --
0, 1, 2, 3
4, 5, 6, 7
8, 9, 10, 11
```

```
12, 13, 14, 15
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GE - Elementwise Greater Than or Equal

This procedure implements an elementwise computation of thee C := A >= B comparison, where A, B, and C are matrices.

Usage

The GE stored procedure has the following syntax:

- **GE(matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same number of dimensions, that is the same number of rows and columns. The output matrix C is given the same shape. Matrix C must not exist prior to the operation. C is a matrix containing only zeros and ones, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..SHAPE('1,15,5,7', 3, 3, 'B');
CALL nzm..GE('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 GE
----
 t
(1 row)
                         PRINT
----------------------------------------
 -- matrix: C --
1, 0, 0
0, 1, 0
1, 1, 1
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GEMM - General Matrix Multiplication

This procedure computes the general matrix multiplication C = A B, where A, B, and C are matrices.

Usage

The GEMM stored procedure has the following syntax:

- **GEMM(NVARCHAR(ANY) matrixA, BOOLEAN transposeA, NVARCHAR(ANY) matrixB, BOOLEAN transposeB, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of the input matrix A.

      Type: NVARCHAR(ANY)
    - **transposeA**
      Specifies whether matrix A should be transposed for multiplication.

      Type: BOOLEAN
    - **matrixB**
      The name of the input matrix B.

      Type: NVARCHAR(ANY)
    - **transposeB**
      Specifies whether matrix A should be transposed for multiplication.

Type: BOOLEAN

- **matrixC**
  The name of the output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..shape('2,2,2,3,3,3,4,4,4', 3, 3, 'B');
CALL nzm..gemm('A', FALSE,'B', TRUE,'C');
CALL nzm..print('C');
CALL nzm..delete_matrix('A' );
CALL nzm..delete_matrix('B' );
CALL nzm..delete_matrix('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 GEMM
------
 t
(1 row)
                      PRINT
--------------------------------------------------
 -- matrix: C --
12, 18, 24
18, 27, 36
42, 63, 84
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GEMM - General Matrix Multiplication - simplified version

This procedure computes the general matrix multiplication C = A B, where A, B, and C are matrices.

Usage

The GEMM stored procedure has the following syntax:

- **GEMM(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of the input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixB**
      The name of the input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of the output matrix C.

      Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Details

This procedure directly calls the BOOLEAN = nzm..GEMM(NVARCHAR(ANY) matrixA, BOOLEAN transposeA, NVARCHAR(ANY) matrixB, BOOLEAN transposeB, NVARCHAR(ANY) matrixC) GEMM variant with input parameters set to: transposeA = FALSE, transposeB = FALSE

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..shape('2,2,2,3,3,3,4,4,4', 3, 3, 'B');
CALL nzm..gemm('A', 'B', 'C');
CALL nzm..print('C');
CALL nzm..delete_matrix('A' );
CALL nzm..delete_matrix('B' );
CALL nzm..delete_matrix('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 GEMM
------
 t
(1 row)
                      PRINT
--------------------------------------------------
 -- matrix: C --
20, 20, 20
23, 23, 23
65, 65, 65
(1 row)
```

```
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GET_NUM_COLS - Return the Number of Columns of a Matrix

This procedure returns the number of columns in the specified matrix.

Usage

The GET_NUM_COLS stored procedure has the following syntax:

- **GET_NUM_COLS(NVARCHAR(ANY) mat_name);**
  - Parameters
    - **mat_name**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    INT4 the number of columns in the matrix

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..get_num_cols('A');
CALL nzm..delete_matrix('A' );
```

```
 SHAPE
-------
 t
(1 row)
 GET_NUM_COLS
--------------
            3
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GET_NUM_ROWS - Return the Number of Rows of a Matrix

This procedure returns the number of rows in the specified matrix.

Usage

The GET_NUM_ROWS stored procedure has the following syntax:

- **GET_NUM_ROWS(NVARCHAR(ANY) mat_name);**
  - Parameters
    - **mat_name**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    INT4 The number of rows in the matrix.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..get_num_rows('A');
CALL nzm..delete_matrix('A' );
```

```
 SHAPE
-------
 t
(1 row)
 GET_NUM_ROWS
--------------
             3
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GET_VALUE - Return the Value of a Matrix Element

This procedure returns the value of the specified matrix element.

Usage

The GET_VALUE stored procedure has the following syntax:

- **GET_VALUE(NVARCHAR(ANY) mat_name, INT4 inrow, INT4 incol);**
  - Parameters
    - **mat_name**
      The name of the matrix.

      Type: NVARCHAR(ANY)
    - **inrow**
      The row index of the element.

      Type: INT4
    - **incol**
      The column index of the element.

      Type: INT4
  - Returns

DOUBLE The value of the matrix element.

Details

This procedure is intended for use with small numbers of values. For retrieving large numbers of values, use alternate approaches that process data in bulk.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..get_value('A', 2, 3);
CALL nzm..delete_matrix('A' );
 SHAPE
-------
 t
(1 row)
 GET_VALUE
-----------
         0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

GT - Elementwise Greater Than

This procedure implements an elementwise computation of the C := A > B comparison, where A, B, and C are matrices.

Usage

The GT stored procedure has the following syntax:

- **GT(matrixAname, matrixBname, matrixCname);**

- Parameters
  - **matrixAname**
    The name of input matrix A.

    Type: NVARCHAR(ANY)
  - **matrixBname**
    The name of input matrix B.

    Type: NVARCHAR(ANY)
  - **matrixCname**
    The name of output matrix C.

    Type: NVARCHAR(ANY)
- Returns
  BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same number of dimensions, that is the same number of rows and columns. The output matrix C is given the same shape. Matrix C must not exist prior to the operation. C is a matrix containing only zeros and ones, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..SHAPE('1,15,5,7', 3, 3, 'B');
CALL nzm..GT('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
```

```
 GT
----
 t
(1 row)
                   PRINT
-------------------------------------------
 -- matrix: C --
0, 0, 0
0, 1, 0
1, 0, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

INITIALIZE - Initializes nzMatrix

This procedure initializes or re-initializes nzMatrix.

Usage

The INITIALIZE stored procedure has the following syntax:

- **INITIALIZE();**

- Returns
  BOOLEAN TRUE always.

Details

This procedure creates or re-creates the shared nzMatrix metadata table in the current database.

Examples

```
CALL nzm..INITIALIZE();
  INITIALIZE
------------
 t
(1 row)
```

Related Functions

- category matrix operations

INSERT - Insert One Matrix into Another

This procedure inserts one matrix into another.

Usage

The INSERT stored procedure has the following syntax:

- **INSERT(matrixIn1, matrixIn2, row_start, col_start)**
  - Parameters
    - **matrixIn1**
      The name of the matrix being inserted into.

      Type: NVARCHAR(ANY)
    - **matrixIn2**
      The name of the matrix to be inserted.

      Type: NVARCHAR(ANY)
    - **row_start**
      The row index where insertion should begin.

      Type: INT4
    - **col_start**
      The column index where insertion should begin.

Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Details

The procedure works in place, modifying matrixIn1.

Examples

```
CALL nzm..SHAPE('0', 4, 4, 'A');
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'B');
CALL nzm..INSERT('A', 'B', 2, 2);
CALL nzm..PRINT('A');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 INSERT
--------
 t
(1 row)
                                PRINT
-----------------------------------------------------------------
 -- matrix: A --
0, 0, 0, 0
0, 1, 2, 3
0, 4, 5, 0
0, 6, 7, 8
(1 row)
```

```
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

INT_ELEMENTS - Elementwise Truncate Function

This procedure implements an elementwise truncating of values for the specified block of elements.

Usage

The INT_ELEMENTS stored procedure has the following syntax:

- **INT_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

Type: INT4

- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Details

The procedure truncates values toward zero. The last four arguments may be omitted, in which case the procedure applies to the entire input matrix.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,11.11,12
.12,13.13,14.14,15.15,16.16',4,4,'A');
CALL nzm..INT_ELEMENTS('A', 'B', 1, 1, 3, 3);
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 INT_ELEMENTS
--------------
 t
(1 row)
                                        PRINT
---------------------------------------------------------------
-----------------------
 -- matrix: B --
0, 1, 2, 3.3
4, 5, 6, 7.7
8, 9, 10, 11.11
12.12, 13.13, 14.14, 15.15
(1 row)
```

```
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

INT_ELEMENTS - Elementwise Truncate Function (entire matrix operation)

This procedure implements an elementwise truncating of values.

Usage

The INT_ELEMENTS stored procedure has the following syntax:

- **INT_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

The procedure truncates values toward zero. The procedure applies to the entire input matrix.

Examples

```
CALL
nzm..SHAPE('0,1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.10,1
```

```
1.11,12.12,13.13,14.14,15.15,16.16',4,4,'A');
CALL nzm..INT_ELEMENTS('A', 'B');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 INT_ELEMENTS
--------------
 t
(1 row)
                               PRINT
---------------------------------------------------------------
----
 -- matrix: B --
0, 1, 2, 3
4, 5, 6, 7
8, 9, 10, 11
12, 13, 14, 15
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

INVERSE - Matrix Inversion

This procedure computes C = inverse(A), where A and C are matrices.

Usage

The INVERSE stored procedure has the following syntax:

- **INVERSE(matrixA, matrixC);**
  - Parameters
    - **matrixA**
      The name of imput matrix A.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..CREATE_RANDOM_MATRIX('A6K', 6000, 6000);
CALL nzm..GEMM_LARGE('A6K', FALSE,'A6K', TRUE,'L6K');
CALL nzm..INVERSE('L6K', 'I6K');
CALL nzm..INVERSE_SMALL('L6K', 'S6K');
CALL nzm..DELETE_MATRIX('A6K' );
CALL nzm..DELETE_MATRIX('L6K' );
CALL nzm..DELETE_MATRIX('I6K' );
CALL nzm..DELETE_MATRIX('S6K' );
CALL nzm..CREATE_RANDOM_MATRIX('A5K', 5000, 5000);
CALL nzm..GEMM_LARGE('A5K', FALSE,'A5K', TRUE,'L5K');
CALL nzm..INVERSE('L5K', 'I5K');
CALL nzm..INVERSE_SMALL('L5K', 'S5K');
CALL nzm..DELETE_MATRIX('A5K' );
CALL nzm..DELETE_MATRIX('L5K' );
```

```
CALL nzm..DELETE_MATRIX('I5K' );
CALL nzm..DELETE_MATRIX('S5K' );
CALL nzm..CREATE_RANDOM_MATRIX('A4K', 4000, 4000);
CALL nzm..GEMM_LARGE('A4K', FALSE,'A4K', TRUE,'L4K');
CALL nzm..INVERSE('L4K', 'I4K');
CALL nzm..INVERSE_SMALL('L4K', 'S4K');
CALL nzm..DELETE_MATRIX('A4K' );
CALL nzm..DELETE_MATRIX('L4K' );
CALL nzm..DELETE_MATRIX('I4K' );
CALL nzm..DELETE_MATRIX('S4K' );
CALL nzm..CREATE_RANDOM_MATRIX('A3K', 3000, 3000);
CALL nzm..GEMM_LARGE('A3K', FALSE,'A3K', TRUE,'L3K');
CALL nzm..INVERSE('L3K', 'I3K');
CALL nzm..INVERSE_SMALL('L3K', 'S3K');
CALL nzm..DELETE_MATRIX('A2K' );
CALL nzm..DELETE_MATRIX('L2K' );
CALL nzm..DELETE_MATRIX('I2K' );
CALL nzm..DELETE_MATRIX('S2K' );
CALL nzm..CREATE_RANDOM_MATRIX('A2K', 2000, 2000);
CALL nzm..GEMM_LARGE('A2K', FALSE,'A2K', TRUE,'L2K');
CALL nzm..INVERSE('L2K', 'I2K');
CALL nzm..INVERSE_SMALL('L2K', 'S2K');
CALL nzm..DELETE_MATRIX('A2K' );
CALL nzm..DELETE_MATRIX('L2K' );
CALL nzm..DELETE_MATRIX('I2K' );
CALL nzm..DELETE_MATRIX('S2K' );
CALL nzm..CREATE_RANDOM_MATRIX('A15K', 1500, 1500);
CALL nzm..GEMM_LARGE('A15K', FALSE,'A15K', TRUE,'L15K');
CALL nzm..INVERSE('L15K', 'I15K');
CALL nzm..INVERSE_SMALL('L15K', 'S15K');
CALL nzm..DELETE_MATRIX('A15K' );
CALL nzm..DELETE_MATRIX('L15K' );
CALL nzm..DELETE_MATRIX('I15K' );
```

```
CALL nzm..DELETE_MATRIX('S15K' );
CALL nzm..CREATE_RANDOM_MATRIX('A1K', 1000, 1000);
CALL nzm..GEMM_LARGE('A1K', FALSE,'A1K', TRUE,'L1K');
CALL nzm..INVERSE('L1K', 'I1K');
CALL nzm..INVERSE_SMALL('L1K', 'S1K');
CALL nzm..DELETE_MATRIX('A1K' );
CALL nzm..DELETE_MATRIX('L1K' );
CALL nzm..DELETE_MATRIX('I1K' );
CALL nzm..DELETE_MATRIX('S1K' );
CALL nzm..CREATE_RANDOM_MATRIX('A05K', 500, 500);
CALL nzm..GEMM_LARGE('A05K', FALSE,'A05K', TRUE,'L05K');
CALL nzm..INVERSE('L05K', 'I05K');
CALL nzm..INVERSE_SMALL('L05K', 'S05K');
CALL nzm..DELETE_MATRIX('A05K' );
CALL nzm..DELETE_MATRIX('L05K' );
CALL nzm..DELETE_MATRIX('I05K' );
CALL nzm..DELETE_MATRIX('S05K' );
CALL nzm..CREATE_RANDOM_MATRIX('A25K', 250, 250);
CALL nzm..GEMM_LARGE('A25K', FALSE,'A25K', TRUE,'L25K');
CALL nzm..INVERSE('L25K', 'I25K');
CALL nzm..INVERSE_SMALL('L25K', 'S25K');
CALL nzm..DELETE_MATRIX('A25K' );
CALL nzm..DELETE_MATRIX('L25K' );
CALL nzm..DELETE_MATRIX('I25K' );
CALL nzm..DELETE_MATRIX('S25K' );
CALL nzm..CREATE_RANDOM_MATRIX('A10K', 100, 100);
CALL nzm..GEMM_LARGE('A10K', FALSE,'A10K', TRUE,'L10K');
CALL nzm..INVERSE('L10K', 'I10K');
CALL nzm..INVERSE_SMALL('L10K', 'S10K');
CALL nzm..DELETE_MATRIX('A10K' );
CALL nzm..DELETE_MATRIX('L10K' );
```

```
CALL nzm..DELETE_MATRIX('I10K' );
CALL nzm..DELETE_MATRIX('S10K' );
CALL nzm..CREATE_RANDOM_MATRIX('A10K', 10, 10);
CALL nzm..GEMM_LARGE('A10K', FALSE,'A10K', TRUE,'L10K');
CALL nzm..INVERSE('L10K', 'I10K');
CALL nzm..INVERSE_SMALL('L10K', 'S10K');
CALL nzm..DELETE_MATRIX('A10K' );
CALL nzm..DELETE_MATRIX('L10K' );
CALL nzm..DELETE_MATRIX('I10K' );
CALL nzm..DELETE_MATRIX('S10K' );
```

```
 CREATE_ONES_MATRIX
--------------------
 t
(1 row)
 INVERSE
---------
 t
(1 row)

PRINT
-----------------------------------------------------------------
-----------------------------------------------------------------
---------------------------------------------------
 -- matrix: B --
0.11111111111111, 0.11111111111111, 0.11111111111111
0.11111111111111, 0.11111111111111, 0.11111111111111
0.11111111111111, 0.11111111111111, 0.11111111111111
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

IS_INITIALIZED - Is Initialized

This procedure checks if the matrix enviroment is initialized.

Usage

The IS_INITIALIZED stored procedure has the following syntax:

- **IS_INITIALIZED();**
  - Returns
    BOOLEAN TRUE if the matrix environment is initialized; FALSE otherwise.

Examples

```
CALL nzm..IS_INITIALIZED();

 IS_INITIALIZED
----------------
 t
(1 row)
```

Related Functions

- category matrix operations

KILL_ENGINE - Kill the Matrix Engine

This procedure kills the Matrix Engine.

Usage

The KILL_ENGINE stored procedure has the following syntax:

- **KILL_ENGINE(engineID);**
  - Parameters
    - **engineID**
      The ID of the engine to be killed.

      Type: INT8
  - Returns
    INT Returns 0 on success.

Details

This procedure is used to kill the Matrix Engine. It can be used to abort a long-running computation, to clean up processes after an error has occurred, or to remove failed jobs from the queue.

Examples

```
CALL nzm..KILL_ENGINE(123456789);

 KILL_ENGINE
-------------
           0
(1 row)
```

Related Functions

- category matrix operations

KRONECKER - Kronecker Product

This procedure computes the Kronecker product of two matrices.

Usage

The KRONECKER stored procedure has the following syntax:

- **KRONECKER(matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)

- **matrixCname**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

Matrices A and B DO NOT need to have the same dimensions, that is, number of rows and columns. The resulting matrix C has dimensions corresponding to the products of the respective dimensions of A and B. Matrix C must not exist prior to the operation. If A is an m by n matrix and B is a k by l matrix, then the Kronecker product m * k by n * l matrix such that C_{i * k + r, j * l + s} = A_{i, j} * B_{r, s}.

Examples

```
CALL nzm..SHAPE('1,10,1000,10000', 2, 2, 'A');
CALL nzm..SHAPE('2,5,7,19', 2, 2, 'B');
CALL nzm..KRONECKER('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 KRONECKER
-----------
 t
(1 row)
                                                     PRINT
```

```
-----------------------------------------------------------------
-----------------------------------
 -- matrix: C --
2, 5, 20, 50
7, 19, 70, 190
2000, 5000, 20000, 50000
7000, 19000, 70000, 190000
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LE - Elementwise less than or equal

This procedure implements an elementwise computation of the C := A <= B comparison, where A, B, and C are matrices.

Usage

The LE stored procedure has the following syntax:

- **LE(matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

Type: NVARCHAR(ANY)

- **matrixBname**
  The name of input matrix B.

  Type: NVARCHAR(ANY)

- **matrixCname**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same number of dimensions, that is, the same number of rows and columns. The output matrix C is given the same shape. Matrix C must not exist prior to the operation. C is a matrix containing only zeros and ones, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..SHAPE('1,15,5,7', 3, 3, 'B');
CALL nzm..LE('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 LE
----
```

```
 t
(1 row)
                    PRINT
-----------------------------------------
 -- matrix: C --
1, 1, 1
1, 0, 1
0, 1, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LINEAR_COMBINATION - Linear Combination of Matrix Components

This procedure implements linear combination of matrix components, computing

matC:=aVal*matA^transposeA + bVal*matB^transposeB + cVal,

where:

matA,matB - input matrices

matC - output matrix

aVal, bVal, cVal - coefficients

transposeA, transposeB - boolean parameters indicating whether matA and matB should be transposed during the operation.

Usage

The LINEAR_COMBINATION stored procedure has the following syntax:

- **LINEAR_COMBINATION(matrixA, transposeA, aValue, matrixB, transposeB, bValue, cValue, matrixC);**
  - Parameters
    - **matrixA**
      The name of the input matrix A.

      Type: NVARCHAR(ANY)
    - **transposeA**
      Specifies whether matrix A must be transposed.

      Type: BOOLEAN
    - **aValue**
      The value of the factor a.

      Type: DOUBLE
    - **matrixB**
      The name of the input matrix A.

      Type: NVARCHAR(ANY)
    - **transposeB**
      Specifies whether matrix A must be transposed.

      Type: BOOLEAN
    - **bValue**
      The value of the factor b.

      Type: DOUBLE
    - **cValue**
      The value of the factor c.

      Type: DOUBLE
    - **matrixC**
      The name of the output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
CALL nzm..create_ones_matrix('A', 4, 4);
CALL nzm..set_value('A', 1, 2, 2);
```

```
CALL nzm..set_value('A', 1, 3, 3);
CALL nzm..set_value('A', 1, 4, 4);
CALL nzm..create_identity_matrix('B', 4);
CALL nzm..set_value('B', 4, 1, 10);
CALL nzm..linear_combination('A', FALSE, 1.5, 'B', FALSE, 1, 1,
'AB');
CALL nzm..linear_combination('A', TRUE, 1.5, 'B', FALSE, 1, 1,
'AtB');
CALL nzm..linear_combination('A', FALSE, 1.5, 'B', TRUE, 1, 1,
'ABt');
CALL nzm..linear_combination('A', TRUE, 1.5, 'B', TRUE, 1, 1,
'AtBt');
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..print('AB');
CALL nzm..print('AtB');
CALL nzm..print('ABt');
CALL nzm..print('AtBt');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
CALL nzm..delete_matrix('AB');
CALL nzm..delete_matrix('AtB');
CALL nzm..delete_matrix('ABt');
CALL nzm..delete_matrix('AtBt');
 CREATE_ONES_MATRIX
--------------------
 t
(1 row)
 SET_VALUE
-----------
 t
(1 row)
 SET_VALUE
-----------
```

```
 t
(1 row)
 SET_VALUE
-----------
 t
(1 row)
 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
 SET_VALUE
-----------
 t
(1 row)
 LINEAR_COMBINATION
--------------------
 t
(1 row)
 LINEAR_COMBINATION
--------------------
 t
(1 row)
 LINEAR_COMBINATION
--------------------
 t
(1 row)
 LINEAR_COMBINATION
--------------------
 t
(1 row)
                           PRINT
```

```
-------------------------------------------------------------
 -- matrix: A --
1, 2, 3, 4
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
(1 row)
                            PRINT
--------------------------------------------------------------
 -- matrix: B --
1, 0, 0, 0
0, 1, 0, 0
0, 0, 1, 0
10, 0, 0, 1
(1 row)
                                           PRINT
---------------------------------------------------------------
----------------------------
 -- matrix: AB --
3.5, 4, 5.5, 7
2.5, 3.5, 2.5, 2.5
2.5, 2.5, 3.5, 2.5
12.5, 2.5, 2.5, 3.5
(1 row)
                                           PRINT
---------------------------------------------------------------
-----------------------------
 -- matrix: AtB --
3.5, 2.5, 2.5, 2.5
4, 3.5, 2.5, 2.5
5.5, 2.5, 3.5, 2.5
17, 2.5, 2.5, 3.5
(1 row)
                                           PRINT
```

```
-----------------------------------------------------------
------------------------------------
 -- matrix: ABt --
3.5, 4, 5.5, 17
2.5, 3.5, 2.5, 2.5
2.5, 2.5, 3.5, 2.5
2.5, 2.5, 2.5, 3.5
(1 row)
                                             PRINT
-----------------------------------------------------------
-------------------------------------
 -- matrix: AtBt --
3.5, 2.5, 2.5, 12.5
4, 3.5, 2.5, 2.5
5.5, 2.5, 3.5, 2.5
7, 2.5, 2.5, 3.5
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LIST_MATRICES - Lists all Matrices in the Connected Database

This procedure lists all matrices in the connected database.

Usage

The LIST_MATRICES stored procedure has the following syntax:

- **LIST_MATRICES();**
  - Returns
    NVARCHAR(ANY) A linefeed-separated (and terminated) string of matrix names.

Details

This procedure returns a linefeed-separated string of matrix names.

Examples

```
CALL nzm..SHAPE('0', 3, 3, 'A');
CALL nzm..SHAPE('1', 3, 3, 'B');
CALL nzm..LIST_MATRICES();
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
 SHAPE
-------
 t
(1 row)
```

```
 SHAPE
-------
 t
(1 row)
 LIST_MATRICES
---------------
 A
B
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LN_ELEMENTS - Elementwise LN Function

This procedure implements an elementwise natural log calculation for the specified block of elements.

Usage

The LN_ELEMENTS stored procedure has the following syntax:

- **LN_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.
      Type: NVARCHAR(ANY)

- **matrixOut**
  The name of the output matrix.

  Type: NVARCHAR(ANY)
- **row_start**
  The first row of the input matrix to use.

  Type: INT4
- **col_start**
  The first column of the input matrix to use.

  Type: INT4
- **row_stop**
  The last row of the input matrix to use.

  Type: INT4
- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

▲ Returns
BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'A');
CALL nzm..LN_ELEMENTS('A', 'B', 2, 2, 3, 3);
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
  SHAPE
-------
 t
(1 row)
 LN_ELEMENTS
-------------
 t
(1 row)
                                                          PRINT
---------------------------------------------------------------
------------------------------------------------------------
```

```
 -- matrix: B --
1, 2, 3, 4
5, 1.7917594692281, 1.9459101490553, 8
9, 2.302585092994, 2.3978952727984, 12
13, 14, 15, 16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LN_ELEMENTS - Elementwise LN Function (entire matrix operation)

This procedure implements an elementwise natural log calculation.

Usage

The LN_ELEMENTS stored procedure has the following syntax:

- **LN_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'A');
CALL nzm..LN_ELEMENTS('A', 'B');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 LN_ELEMENTS
-------------
 t
(1 row)

PRINT
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
------------------
 -- matrix: B --
0, 0.69314718055995, 1.0986122886681, 1.3862943611199
1.6094379124341, 1.7917594692281, 1.9459101490553,
2.0794415416798
2.1972245773362, 2.302585092994, 2.3978952727984,
2.484906649788
2.5649493574615, 2.6390573296153, 2.7080502011022,
2.7725887222398
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LOC - Locate Non-zero Elements

This procedure locates the vector of positions of non-zero elements.

Usage

The LOC stored procedure has the following syntax:

- **LOC(NVARCHAR(ANY) matrixIn, NVARCHAR(ANY) matrixOut);**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

The procedure returns a row vector of indices positioning the non-zero elements of the input matrix. Index values are in row-major order, and the indices must be in the range from 1 to the number of elements in the first argument. If all elements are zero, the result is a NULL, as a matrix with zero rows and zero columns cannot be created, and an error occurs. The statement loc('AA','CC'); for a one row matrix AA={25,0,71,18} returns a row vector {1,3,4}. The output matrix must not exist prior to the operation.

Examples

```
CALL
nzm..SHAPE('0,1,2,3,4,5,6,7,8,0,0,0,0,3,4,5',4,4,'A');

CALL nzm..LOC('A', 'B');
```

```
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 LOC
-----
 t
(1 row)
                      PRINT
---------------------------------------------------
 -- matrix: B --
2, 3, 4, 5, 6, 7, 8, 9, 14, 15, 16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LOG_ELEMENTS - Elementwise log Function of any base

This procedure implements the elementwise log operation of any base.

Usage

The LOG_ELEMENTS stored procedure has the following syntax:

- **LOG_ELEMENTS('matrixIn', 'matrixOut', log_base)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **log_base**
      The base to use for the log operation.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
A');

CALL nzm..LOG_ELEMENTS('A', 'B', 3);

CALL nzm..PRINT('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 LOG_ELEMENTS
--------------
 t
(1 row)
```

```
PRINT
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 -- matrix: B --
0, 0.63092975357146, 1, 1.2618595071429
1.4649735207179, 1.6309297535715, 1.7712437491614,
1.8927892607144
2, 2.0959032742894, 2.1826583386441, 2.2618595071429
2.3347175194728, 2.4021735027329, 2.4649735207179,
2.5237190142858
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LOG_ELEMENTS - Elementwise log Function of any base for the specified block of elements

This procedure implements the elementwise log operation of any base for the specified block of elements.

Usage

The LOG_ELEMENTS stored procedure has the following syntax:

- **LOG_ELEMENTS('matrixIn', 'matrixOut', log_base, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

Type: NVARCHAR(ANY)

- **matrixOut**
  The name of the output matrix.

  Type: NVARCHAR(ANY)

- **log_base**
  The base to use for the log operation.

  Type: INT4

- **row_start**
  The first row of the input matrix to use.

  Type: INT4

- **col_start**
  The first column of the input matrix to use.

  Type: INT4

- **row_stop**
  The last row of the input matrix to use.

  Type: INT4

- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
A');

CALL nzm..LOG_ELEMENTS('A', 'B', 3 , 2, 2, 3, 3);

CALL nzm..PRINT('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 LOG_ELEMENTS
```

```
--------------
 t
(1 row)
                                                             PRINT
-------------------------------------------------------------
---------------------------------------------------------
 -- matrix: B --
1, 2, 3, 4
5, 1.6309297535715, 1.7712437491614, 8
9, 2.0959032742894, 2.1826583386441, 12
13, 14, 15, 16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LOG_ELEMENTS - Elementwise log Function of base 10

This procedure implements the elementwise log operation of base 10.

Usage

The LOG_ELEMENTS stored procedure has the following syntax:

- **LOG_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.
      Type: NVARCHAR(ANY)

- ► **matrixOut**
  The name of the output matrix.

  Type: NVARCHAR(ANY)

- ▲ Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
A');

CALL nzm..LOG_ELEMENTS('A', 'B');

CALL nzm..PRINT('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 LOG_ELEMENTS
--------------
 t
(1 row)

PRINT
---------------------------------------------------------
---------------------------------------------------------
---------------------------------------------------------
---------------------------------------------------------
-------------------------------------
 -- matrix: B --
0, 0.30102999566398, 0.47712125471966, 0.60205999132796
0.69897000433602, 0.77815125038364, 0.84509804001426,
0.90308998699194
0.95424250943932, 1, 1.0413926851582, 1.0791812460476
1.1139433523068, 1.1461280356782, 1.1760912590557,
```

```
1.2041199826559
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

LT - Elementwise Less Than

This procedure implements elementwise computation of the C := A < B comparison, where A, B, and C are matrices.

Usage

The LT stored procedure has the following syntax:

- **LT(matrixAname,matrixBname,matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same number of dimensions, that is, the same number of rows and columns. The output matrix C is given the same shape. Matrix C must not exist prior to the operation. C is a matrix containing only zeros and ones, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..SHAPE('1,15,5,7', 3, 3, 'B');
CALL nzm..LT('A', 'B', 'C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 LT
----
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: C --
0, 1, 1
1, 0, 1
0, 0, 0
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MATRIX_EXISTS - Check if a Matrix Exists

This procedure checks if a matrix with the specified name exists.

Usage

The MATRIX_EXISTS stored procedure has the following syntax:

- **MATRIX_EXISTS(NVARCHAR(ANY) mat_name);**
  - Parameters
    - **mat_name**
      The matrix name.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if the matrix exists.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8', 3, 3, 'A');
CALL nzm..MATRIX_EXISTS('A');
CALL nzm..DELETE_MATRIX('A' );
 SHAPE
-------
 t
```

```
(1 row)
 MATRIX_EXISTS
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MATRIX_VECTOR_OPERATION - Elementwise Matrix-vector Operation

This procedure implements elementwise matrix-vector operations.

Usage

The MATRIX_VECTOR_OPERATION stored procedure has the following syntax:

- **MATRIX_VECTOR_OPERATION('matrixIn', 'matrixOut', 'vector', 'operator', 'orientation')**
  - Parameters
    - **Input**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **Output**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **Vector**
      The name of the vector matrix.

      Type: NVARCHAR(ANY)
    - **operator**
      The operator to use. Must be one of the following: + - * / % ^ & |

      Type: NVARCHAR(ANY)
    - **Orientation**
      The orientation of the operation, that is, whether it should be applied to

'r' - rows: [Input matrix][i,j] -> [Input matrix][i,j] [operator] [vector][j]

'c' - columns

'd' - diagonal.

Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

The procedure implements elementwise matrix-vector operations. Depending on the specified orientation, each row, column or the diagonal X is transformed in the form X_new:=X [operator] 'vector'.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9', 3, 3, 'A');
CALL nzm..REDUCE_TO_VECT('A','V','AVG',null,'r');
CALL nzm..MATRIX_VECTOR_OPERATION('A', 'B', 'V', '-','r');
CALL nzm..PRINT('A');
CALL nzm..PRINT('V');
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('V' );
```

```
 SHAPE
-------
 t
(1 row)
 REDUCE_TO_VECT
----------------
 t
(1 row)
 MATRIX_VECTOR_OPERATION
-------------------------
 t
(1 row)
                    PRINT
```

```
-------------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 6
7, 8, 9
(1 row)
          PRINT
--------------------------
 -- matrix: V --
4, 5, 6
(1 row)
                   PRINT
----------------------------------------------
 -- matrix: B --
-3, -3, -3
0, 0, 0
3, 3, 3
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MAX - Elementwise Maximum, Elementwise Logical OR

This procedure implements an elementwise computation of C := max(A, B), where A, B, and C are matrices.

Usage

The MAX stored procedure has the following syntax:

- **MAX(matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

If matrices A and B are logical matrices consisting of zeros (0) as FALSE and ones (1) as TRUE, then C := A | B (elementwise "OR"). Matrices A and B must have the same dimensions, that is, the same number of rows and columns. Matrix C is given the same shape. Matrix C must not exist prior to the operation

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9', 3, 3, 'A');
CALL nzm..SHAPE('9,8,7,6,5,4,3,2,1', 3, 3, 'B');
CALL nzm..MAX('A','B','C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
 SHAPE
-------
```

```
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 MAX
-----
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: C --
9, 8, 7
6, 5, 6
7, 8, 9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MIN - Elementwise Minimum, Elementwise Logical AND

This procedure implements an elementwise computation of C := min(A, B), where A, B, and C are matrices.

Usage

The MIN stored procedure has the following syntax:

- **MIN(matrixAname, matrixBname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

If matrices A and B are logical matrices consisting of zeros (0) as FALSE and ones (1) as TRUE, then C := A | B (elementwise "AND"). Matrices A and B must have the same dimensions, that is, the same number of rows and columns. Matrix C is given the same shape. Matrix C must not exist prior to the operation

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9', 3, 3, 'A');
CALL nzm..SHAPE('9,8,7,6,5,4,3,2,1', 3, 3, 'B');
CALL nzm..MIN('A','B','C');
CALL nzm..PRINT('C');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
CALL nzm..DELETE_MATRIX('C' );
 SHAPE
-------
```

```
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 MIN
-----
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: C --
1, 2, 3
4, 5, 4
3, 2, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MOD_ELEMENTS - Elementwise MOD Function

This function implements the elementwise modulo operation for the specified block of elements.

Usage

The MOD_ELEMENTS stored procedure has the following syntax:

- **MOD_ELEMENTS('matrixIn','matrixOut',divisor, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **divisor**
      The divisor.

      Type: DOUBLE
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9', 3, 3, 'A');
CALL nzm..MOD_ELEMENTS('A', 'B', 3 , 2, 2, 2, 2);
```

```
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 MOD_ELEMENTS
--------------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: B --
1, 2, 3
4, 2, 6
7, 8, 9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MOD_ELEMENTS - Elementwise MOD Function (entire matrix operation)

This procedure implements an elementwise modulo operation.

Usage

The MOD_ELEMENTS stored procedure has the following syntax:

- **MOD_ELEMENTS('matrixIn','matrixOut',divisor)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **divisor**
      The divisor to use.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,6,7,8,9', 3, 3, 'A');
CALL nzm..MOD_ELEMENTS('A', 'B', 3 );
CALL nzm..PRINT('B');
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 MOD_ELEMENTS
--------------
 t
(1 row)
                      PRINT
```

```
-----------------------------------------
 -- matrix: B --
1, 2, 0
1, 2, 0
1, 2, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_LINEAR_REGRESSION - Linear Regression

This procedure creates the linear regression model using data stored in a matrix.

Usage

The MTX_LINEAR_REGRESSION stored procedure has the following syntax:

- **MTX_LINEAR_REGRESSION(NVARCHAR(ANY) modelName, NVARCHAR(ANY) predictorsMatrixName, NVARCHAR(ANY) predictedMatrixName, BOOLEAN includeIntercept, BOOLEAN calculateDiagnostics, BOOLEAN useSVDSolver);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **predictorsMatrixName**
      The name of the matrix containing the predictors.

      Type: NVARCHAR(ANY)

- **predictedMatrixName**
  The name of the matrix containing predicted values.

  Type: NVARCHAR(ANY)
- **includeIntercept**
  Specified whether the intercept term should be included in the model.

  Type: BOOLEAN
- **calculateDiagnostics**
  Specified whether diagnostics information should be provided.

  Type: BOOLEAN
- **useSVDSolver**
  Specifies whether Singular Value Decomposition and matrix multiplication should be used for solving the matrix equation.

  Type: BOOLEAN

- Returns
  BOOLEAN TRUE only if the diagnostical information has been generated, for example, in the case of calculateDiagnostics=TRUE and number of model parameters larger than number of observations.

Details

This procedure builds the linear regression model using the QR solver of a non-singular model matrix, or the Moore-Penrose pseudoinversion in the case of a near-singular or exactly singular model matrix. Input data should be provided as Database Matrix Objects with observations provided in rows, and predictors in columns. The matrix of predicted values may contain multiple columns, that is, multiple predicted values. The diagnostic information, if requested, is saved as a set of matrices of names starting with modelName_linearmodel prefix. The set consists of following matrices:

modelName_linearmodel_R2 - row vector containing R^2 (being a fraction of variance explained by the model) of models created for each output attribute (when calculateDiagnostics is TRUE)

modelName_linearmodel_RSS - row vector containing Residual Sum of Squares of models created for each output attribute (when calculateDiagnostics is TRUE)

modelName_linearmodel_SDEV - the matrix of standard deviations of model coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetermined)

modelName_linearmodel_TVAL - the matrix of the test statistics for the models' coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetermined)

modelName_linearmodel_PVAL - the matrix of the two--sided p-values for the models' coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetermined)

modelName_linearmodel_Y_VAR_EST - the row vector containing the estimators of a variance of error term for each predicted variable (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetermined)

Model coefficients are saved as the matrix named modelName_linearmodel.

The constructed model can be applied to the data using the MTX_LINEAR_REGRESSION_APPLY procedure. Note that use of the Singular Value Decomposition and matrix multiplication is slower than the standard calculation, but is more stable in the case of an ill-posed, that is, near colinear, regression model.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9', 100, 10,
'LR_EXAMPLE');

call nzm..shape('9,8,7,6,5,4,3,2,1', 10, 1,
'LR_EXAMPLE_TRUE_COEFFS');

call nzm..gemm('LR_EXAMPLE', 'LR_EXAMPLE_TRUE_COEFFS',
'LR_EXAMPLE_PREDICTED');

call
nzm..mtx_linear_regression('LR_EXAMPLE_MODEL','LR_EXAMPLE
', 'LR_EXAMPLE_PREDICTED', FALSE, FALSE, FALSE);

--- result verification

call nzm..copy_submatrix('LR_EXAMPLE_MODEL_linearmodel',
'LR_EXAMPLE_MODEL_linearmodel_eff', 1, 10, 1, 1);

call nzm..subtract('LR_EXAMPLE_TRUE_COEFFS',
'LR_EXAMPLE_MODEL_linearmodel_eff',
'LR_EXAMPLE_MODEL_verif1');

call nzm..red_max_abs('LR_EXAMPLE_MODEL_verif1');

call nzm..delete_all_matrices();

 SHAPE
-------
 t
(1 row)

 SHAPE
-------
 t
(1 row)

 GEMM
------
 t
(1 row)

 MTX_LINEAR_REGRESSION
-----------------------
 f
(1 row)
```

```
 COPY_SUBMATRIX
----------------
 t
(1 row)
 SUBTRACT
----------
 t
(1 row)
   RED_MAX_ABS
-----------------
 8.3857463735873
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_LINEAR_REGRESSION_APPLY - Linear Regression Model Applier

This procedure applies a linear regression matrix model to data stored in a matrix.

Usage

The MTX_LINEAR_REGRESSION_APPLY stored procedure has the following syntax:

- **MTX_LINEAR_REGRESSION_APPLY(NVARCHAR(ANY) modelName, NVARCHAR(ANY) predictorsMatrixName, NVARCHAR(ANY) predictedMatrixName);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **predictorsMatrixName**
      The name of the matrix containing the predictors.

Type: NVARCHAR(ANY)

- **predictedMatrixName**
  The name of the matrix containing predicted values.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Details

This procedure applies the linear regression model built with the MTX_LINEAR_REGRESSION procedure to the provided data. Input data should be provided as Database Matrix Objects with observations provided in rows, and predictors in columns. The matrix of predicted values may contain multiple columns, that is, multiple predicted values.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9', 100, 10,
'LR_EXAMPLE');

call nzm..shape('9,8,7,6,5,4,3,2,1', 10, 1,
'LR_EXAMPLE_TRUE_COEFFS');

call nzm..gemm('LR_EXAMPLE', 'LR_EXAMPLE_TRUE_COEFFS',
'LR_EXAMPLE_TRUEVAL');

call nzm..mtx_linear_regression('LR_EXAMPLE_MODEL',
'LR_EXAMPLE', 'LR_EXAMPLE_TRUEVAL', FALSE, FALSE, FALSE);

call nzm..mtx_linear_regression_apply('LR_EXAMPLE_MODEL',
'LR_EXAMPLE', 'LR_EXAMPLE_PREDICTED');

call nzm..subtract('LR_EXAMPLE_PREDICTED',
'LR_EXAMPLE_TRUEVAL', 'LR_EXAMPLE_MODEL_verif');

call nzm..red_max_abs('LR_EXAMPLE_MODEL_verif');

call nzm..delete_all_matrices();
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 GEMM
```

```
------
 t
(1 row)
 MTX_LINEAR_REGRESSION
-----------------------
 f
(1 row)
 MTX_LINEAR_REGRESSION_APPLY
-----------------------------

(1 row)
 SUBTRACT
----------
 t
(1 row)
     RED_MAX_ABS
---------------------
 1.1368683772162e-13
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_PCA - Principal Component Analysis (PCA)

This procedure performs a Principal Component Analysis (PCA) using data stored in a matrix.

Usage

The MTX_PCA stored procedure has the following syntax:

- **MTX_PCA(NVARCHAR(ANY) modelName, NVARCHAR(ANY) dataMatrixName, BOOLEAN forceSufficientStats, BOOLEAN centerData, BOOLEAN scaleData, BOOLEAN saveScores);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **dataMatrixName**
      The name of the matrix containing the data.

      Type: NVARCHAR(ANY)
    - **forceSufficientStats**
      Specifies whether the PCA should be based on a covariance matrix even if SVD can be performed.

      Type: BOOLEAN

      Default: FALSE
    - **centerData**
      Specifies whether the model should include data centering, that is, subtraction of the mean estimator.

      Type: BOOLEAN

      Default: TRUE
    - **scaleData**
      Specifies whether the model should include data scaling, which is division by a non-zero standard deviation estimator. When data scaling is performed the resulting PCA model is equivalent to a model based on the correlation matrix

      Type: BOOLEAN

      Default: TRUE
    - **saveScores**
      Specifies whether the PCA scores of individual observations are to be saved.

      Type: BOOLEAN

      Default: FALSE
  - Returns
    BOOLEAN TRUE always.

Details

This procedure constructs a PCA model of the data and provides a corresponding transformation into principal components, which can then be applied using MTX_PCA_APPLY. Input data should be provided as Database Matrix Objects, with observations provided in rows, and attributes in columns.

The PCA can be constructed using two strategies: SVD decomposition, which is more accurate but at the expense of speed and memory, or by finding the eigenvectors of the unbiased covariance matrix estimator. If the parameter forceSufficientStats is not TRUE, the best strategy, that is, the one providing the most accurate solution based on data size and memory availability, is used. Based on the specified parameters, the data matrix can be centered and scaled. In that case the corresponding parameters, the mean and variance estimators are calculated and become part of the model. When included in the model, centering and scaling is also performed during the application step.

Data centering (assuring that mean of each attribute is equal to 0) is an assumption of PCA method - failing to meet it usually causes serious model degradation. Data scaling (assuring that the variance of each attribute is equal to 1) usually provides better approximation of the data in case of the presence of attributes that differ in orders of magnitude. It is equivalent to perform the PCA using the correlation instead of covariance matrix.

In order to express the model being created, the procedure creates a set of matrices, using the modelName parameter as the prefix for given matrix name. The set consists of following matrices:

{prefix}_PCA_ATTMEAN - row vector containing mean values of the attributes (when centerData is TRUE)

{prefix}_PCA_ATTSD - row vector containing standard deviations of the attributes (when scaleData is TRUE)

{prefix}_PCA_ATTSD_DIV - row vector containing reciprocals of non-zero standard deviations of the attributes or value 1 (when scaleData is TRUE)

{prefix}_PCA_SDEV - row vector containing standard deviations of the principal components

{prefix}_PCA - the matrix of loadings (a matrix whose columns contain the eigenvectors of the covariance matrix)

{prefix}_PCA_SCORES - the matrix of scores containing projections of individual obervations to principal components (when saveScores is TRUE)

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9', 1, 3, 'PCA_TEST');

call nzm..shape('9,8,7,6,5,4,3,2,1', 10, 1,
'PCA_TEST_SOURCE_PRE');

--- expected value is 0.0

call
nzm..SCALAR_OPERATION('PCA_TEST_SOURCE_PRE','PCA_TEST_SOURCE',
'-', 0.5);

call nzm..gemm('PCA_TEST_SOURCE', 'PCA_TEST', 'PCA_TEST_VALS');

call nzm..mtx_pca('PCA_TEST_MOD', 'PCA_TEST_VALS', FALSE, FALSE,
FALSE, TRUE);

call nzm..list_matrices();

--- std dev in each direction (in this example real value of all
components other than the first one should be 0)

call nzm..print('PCA_TEST_MOD_PCA_SDEV');

call nzm..print('PCA_TEST_MOD_PCA_SCORES');

--- projecting on the original value (first column)
```

```
call nzm..gemm_large('PCA_TEST_VALS', FALSE,
'PCA_TEST_MOD_PCA', FALSE, 'PCA_TEST_PROJ');

--- resulting value (first column of PCA_TEST_PROJ) is
proportional to original one (PCA_TEST_VALS):
PCA_TEST_PROJ[1,] ~~PCA_TEST_SOURCE *
sqrt(nzm..red_ssq('PCA_TEST'))

call nzm..delete_matrix_like('PCA\_TEST%');

 SHAPE

-------

 t

(1 row)

 SHAPE

-------

 t

(1 row)

 SCALAR_OPERATION

------------------

 t

(1 row)

 GEMM

------

 t

(1 row)

 MTX_PCA

---------

 t

(1 row)

                                                       LI
ST_MATRICES

----------------------------------------------------------
----------------------------------------------------------
----------

 PCA_TEST

PCA_TEST_MOD_PCA
```

```
PCA_TEST_MOD_PCA_SCORES
PCA_TEST_MOD_PCA_SDEV
PCA_TEST_SOURCE
PCA_TEST_SOURCE_PRE
PCA_TEST_VALS
(1 row)

                                            PRINT
-----------------------------------------------------------------
------------------------------
 -- matrix: PCA_TEST_MOD_PCA_SDEV --
22.118368434905
2.4603199788269e-16
9.9446202776076e-17
(1 row)

PRINT
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
-----
 -- matrix: PCA_TEST_MOD_PCA_SCORES --
-31.804087787578, -1.4567015001103e-16, 1.2617124776816e-16
-28.062430400805, -4.1645192033268e-17, -2.6226789760277e-16
-24.320773014031, -3.6092499762165e-17, 3.1261282628732e-17
-20.579115627257, -3.0539807491063e-17, 2.6451854532004e-17
-16.837458240483, -2.4987115219961e-17, 2.1642426435276e-17
-13.095800853709, 7.1866157069922e-16, 1.6832998338548e-17
-9.3541434669349, -1.3881730677756e-17, 1.202357024182e-17
-5.6124860801609, -8.3290384066535e-18, 7.2141421450919e-18
-1.870828693387, -2.7763461355512e-18, 2.404714048364e-18
```

```
-31.804087787578, -4.719788430437e-17, 4.0880138822188e-
17
(1 row)
 GEMM_LARGE
------------
 t
(1 row)
 DELETE_MATRIX_LIKE
--------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_PCA - Principal Component Analysis (PCA) - Non-storing Individual Observations Version

This procedure performs a Principal Component Analysis (PCA) using data stored in a matrix.

Usage

The MTX_PCA stored procedure has the following syntax:

- **MTX_PCA(NVARCHAR(ANY) modelName, NVARCHAR(ANY) dataMatrixName, BOOLEAN forceSufficientStats, BOOLEAN centerData, BOOLEAN scaleData);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **dataMatrixName**
      The name of the matrix containing the data.

      Type: NVARCHAR(ANY)
    - **forceSufficientStats**
      Specifies whether the PCA should be based on a covariance matrix even if SVD can be performed.

Type: BOOLEAN

Default: FALSE

- **centerData**
  Specifies whether the model should include data centering, that is, subtraction of the mean estimator.

  Type: BOOLEAN

  Default: TRUE

- **scaleData**
  Specifies whether the model should include data scaling, which is division by a non-zero standard deviation estimator. When data scaling is performed the resulting PCA model is equivalent to a model based on the correlation matrix.

  Type: BOOLEAN

  Default: TRUE

- Returns
  BOOLEAN Always returns TRUE.

Details

This procedure directly calls the BOOLEAN = nzm..mtx_PCA(NVARCHAR(ANY) modelName, NVARCHAR(ANY) dataMatrixName, BOOLEAN forceSufficientStats, BOOLEAN centerData, BOOLEAN scaleData, BOOLEAN saveScores) PCA variant with the saveScores input parameter set to FALSE.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9', 1, 3, 'PCA_TEST');

call nzm..shape('9,8,7,6,5,4,3,2,1', 10, 1,
'PCA_TEST_SOURCE_PRE');

--- expected value is 0.0

call
nzm..SCALAR_OPERATION('PCA_TEST_SOURCE_PRE','PCA_TEST_SOURCE',
'-', 0.5);

call nzm..gemm('PCA_TEST_SOURCE', 'PCA_TEST', 'PCA_TEST_VALS');

call nzm..mtx_pca('PCA_TEST_MOD', 'PCA_TEST_VALS', FALSE, FALSE,
FALSE);

call nzm..list_matrices();

--- std dev in each direction (in this example real value of all
components other than the first one should be 0)

call nzm..print('PCA_TEST_MOD_PCA_SDEV');

--- projecting on the original value (first column)

call nzm..gemm_large('PCA_TEST_VALS', FALSE, 'PCA_TEST_MOD_PCA',
FALSE, 'PCA_TEST_PROJ');
```

```
--- resulting value (first column of PCA_TEST_PROJ) is
proportional to original one (PCA_TEST_VALS):
PCA_TEST_PROJ[1,] ~~PCA_TEST_SOURCE *
sqrt(nzm..red_ssq('PCA_TEST'))

call nzm..delete_all_matrices();
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 SCALAR_OPERATION
------------------
 t
(1 row)
 GEMM
------
 t
(1 row)
 MTX_PCA
---------
 t
(1 row)
                                           LIST_MATRICES
----------------------------------------------------------
-------------------------------------------
 PCA_TEST
PCA_TEST_MOD_PCA
PCA_TEST_MOD_PCA_SDEV
PCA_TEST_SOURCE
PCA_TEST_SOURCE_PRE
```

```
PCA_TEST_VALS
(1 row)
                                                      PRINT
------------------------------------------------------------------------------
------------------------------
 -- matrix: PCA_TEST_MOD_PCA_SDEV --
22.118368434905
1.4823621419932e-15
4.2901584776278e-16
(1 row)
 GEMM_LARGE
------------
 t
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_PCA - Principal Component Analysis (PCA) - Simplified Version

This procedure performs a Principal Component Analysis (PCA) using data stored in a matrix.

Usage

The MTX_PCA stored procedure has the following syntax:

- **MTX_PCA(NVARCHAR(ANY) modelName, NVARCHAR(ANY) dataMatrixName);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **dataMatrixName**
      The name of the matrix containing the data.

Type: NVARCHAR(ANY)

- Returns
  BOOLEAN Always returns TRUE.

Details

This procedure directly calls the BOOLEAN = nzm..mtx_PCA(NVARCHAR(ANY) modelName, NVARCHAR(ANY) dataMatrixName, BOOLEAN forceSufficientStats, BOOLEAN centerData, BOOLEAN scaleData, BOOLEAN saveScores) PCA variant with input parameters set to: forceSufficientStats = FALSE, centerData = TRUE, scaleData = TRUE, saveScores = FALSE.

Examples

```
call nzm..shape('1,10', 1, 3, 'PCA_TEST');

call nzm..shape('10,20', 10, 1, 'PCA_TEST_SOURCE_PRE');

--- expected value is 0.0

call
nzm..SCALAR_OPERATION('PCA_TEST_SOURCE_PRE','PCA_TEST_SOU
RCE', '-', 0.5);

call nzm..gemm('PCA_TEST_SOURCE', 'PCA_TEST',
'PCA_TEST_VALS');

call nzm..mtx_pca('PCA_TEST_MOD', 'PCA_TEST_VALS');

--- std dev in each direction (in this example real value
of all components other than the first one should be 0)

call nzm..print('PCA_TEST_MOD_PCA_SDEV');

--- projecting on the original value (first column)

call nzm..gemm_large('PCA_TEST_VALS', FALSE,
'PCA_TEST_MOD_PCA', FALSE, 'PCA_TEST_PROJ');

--- resulting value (first column of PCA_TEST_PROJ) is
proportional to original one (PCA_TEST_VALS):
PCA_TEST_PROJ[1,] ~~PCA_TEST_SOURCE *
sqrt(nzm..red_ssq('PCA_TEST'))

call nzm..delete_all_matrices();
```

```
  SHAPE
-------
 t
(1 row)
 SHAPE
-------
```

```
 t
(1 row)
 SCALAR_OPERATION
------------------
 t
(1 row)
 GEMM
------
 t
(1 row)
 MTX_PCA
---------
 t
(1 row)
                                   PRINT
---------------------------------------------------------------
------------
 -- matrix: PCA_TEST_MOD_PCA_SDEV --
1.7320508075689
3.6259732146947e-16
0
(1 row)
 GEMM_LARGE
------------
 t
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_PCA_APPLY - PCA Model Applier

This procedure applies a PCA matrix model to data stored in a matrix.

Usage

The MTX_PCA_APPLY stored procedure has the following syntax:

- **MTX_PCA_APPLY(NVARCHAR(ANY) modelName, NVARCHAR(ANY) matrixToProject, NVARCHAR(ANY) outputMatrix, INT4 numberOfVectors);**
  - Parameters
    - **modelName**
      The name of the created model.

      Type: NVARCHAR(ANY)
    - **matrixToProject**
      The name of the matrix to be projected using the PCA model.

      Type: NVARCHAR(ANY)
    - **outputMatrix**
      The name of the matrix in which to store the result.

      Type: NVARCHAR(ANY)
    - **numberOfVectors**
      The number of principal components used in projection.

      Type: INT4
  - Returns
    BOOLEAN TRUE always.

Details

This procedure applies a PCA transformation constructed using PCA to the provided Database Matrix Object. Each row of the provided matrix is projected on the number of principal components, specified by the numberOfVectors parameter. If an applied model was constructed with centering and scaling operations, the corresponding operations are performed on the provided data using model coefficients based on the original set.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9', 1, 4, 'PCA_TEST');

call nzm..shape('9,8,7,6,5,4,3,2,1', 100, 1,
'PCA_TEST_SOURCE');

call nzm..gemm('PCA_TEST_SOURCE', 'PCA_TEST',
'PCA_TEST_VALS');
```

```
call nzm..mtx_pca('PCA_TEST_MOD', 'PCA_TEST_VALS',  FALSE, TRUE,
TRUE);
--- std dev in each direction (in this example real value of all
components other than the first one should be 0)
call nzm..print('PCA_TEST_MOD_PCA_SDEV');
--- projecting on the original value (first column)
call nzm..mtx_pca_apply('PCA_TEST_MOD', 'PCA_TEST_VALS',
'PCA_TEST_PROJ', 1);
call nzm..delete_all_matrices();
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 GEMM
------
 t
(1 row)
 MTX_PCA
---------
 t
(1 row)
                                               PRINT
---------------------------------------------------------------
------------------------------------
 -- matrix: PCA_TEST_MOD_PCA_SDEV --
2
2.0237717020326e-16
8.1474073981796e-17
7.9324562134617e-33
(1 row)
```

```
 MTX_PCA_APPLY
---------------
 t
(1 row)
 DELETE_ALL_MATRICES
---------------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_POW - nth Power of a Matrix

This procedure multiplies a matrix n times.

Usage

The MTX_POW stored procedure has the following syntax:

- **MTX_POW**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **n**
      The power used to raise the matrix.

      Type: INT4
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

Implements the operation C := A ** n, where n is a natural number and A and C are matrices. Matrix A must be a square matrix. Matrix C must not exist prior to the operation.

NOTE: For larger exponents use the mtx_pow2 procedure, which is optimized for quicker calculation.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..mtx_pow('A',6,'B');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 MTX_POW
---------
 t
(1 row)
                                                  PRINT
-------------------------------------------------------------------------------------------
 -- matrix: B --
488907, 624285, 431775
306552, 391737, 269148
1491562, 1904635, 1316950
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MTX_POW2 - nth Power of a Matrix

This procedure, optimized for larger exponent values, multiplies a matrix n times.

Usage

The MTX_POW2 stored procedure has the following syntax:

- **MTX_POW2(matrixAname,n,matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **n**
      The power used to raise the matrix.

      Type: INT4
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

This procedure, like MTX_POW, implements the operation C := A ** n, where n is a natural number and A and C are matrices. However, this procedure is optimized for larger exponents. Matrix A must be a square matrix. Matrix C must not exist prior to the operation.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..mtx_pow2('A',6,'B');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
SHAPE
```

```
-------
 t
(1 row)
 MTX_POW2
----------
 t
(1 row)
                                            PRINT
-----------------------------------------------------------------
--------------------------
 -- matrix: B --
488907, 624285, 431775
306552, 391737, 269148
1491562, 1904635, 1316950
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

MULTIPLY_ELEMENTS - Multiply Matrices Element-by-element

This procedure computes C, the element-by-element multiplication of A times B: Cij = Aij * Bij.

Usage

The MULTIPLY_ELEMENTS stored procedure has the following syntax:

- **MULTIPLY_ELEMENTS(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) mat-**

rixC);

- Parameters
  - **matrixA**
    The name of input matrix A.

    Type: NVARCHAR(ANY)
  - **matrixB**
    The name of input matrix B.

    Type: NVARCHAR(ANY)
  - **matrixC**
    The name of output matrix C.

    Type: NVARCHAR(ANY)
- Returns
  BOOLEAN TRUE always.

Examples

```
CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
A');

CALL
nzm..SHAPE('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
B');

CALL nzm..MULTIPLY_ELEMENTS('A', 'B', 'C');

CALL nzm..PRINT('B');

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );

CALL nzm..DELETE_MATRIX('C' );

 SHAPE

-------

 t

(1 row)

 SHAPE

-------

 t

(1 row)

 MULTIPLY_ELEMENTS

-------------------
```

```
 t
(1 row)
                                                              PRINT
---------------------------------------------------------------
-----
 -- matrix: B --
1, 2, 3, 4
5, 6, 7, 8
9, 10, 11, 12
13, 14, 15, 16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

NE - Elementwise Not Equal

This procedure implements an elementwise computation of the C := A <> B comparison, where A, B, and C are matrices.

Usage

The NE stored procedure has the following syntax:

- **NE(matrixAname,matrixBname,matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

Matrices A and B must have the same dimensions, that is, the same number of rows and columns. Matrix C is given the same shape. Matrix C must not exist prior to the operation. Matrix C contains only 0 and 1, corresponding to FALSE and TRUE at respective positions.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..shape('1,15,5,7',3,3,'B');
CALL nzm..ne('A','B','C');
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..print('C');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
CALL nzm..delete_matrix('C');
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
```

```
 t
(1 row)
 NE
----
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                   PRINT
-------------------------------------------
 -- matrix: B --
1, 15, 5
7, 1, 15
5, 7, 1
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: C --
0, 1, 1
1, 1, 1
1, 0, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
```

```
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

NORMAL - Matrix of Random, Normally Distributed Values

This procedure creates a new matrix filled with normally distributed random values using drand48_r.

Usage

The NORMAL stored procedure has the following syntax:

- **NORMAL(matrixOut, numberOfRows, numberOfColumns , mean, stddev)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of rows to be included in the created matrix.

      Type: INT4
    - **numberOfColumns**
      The number of columns to be included in the created matrix.

      Type: INT4
    - **mean**
      The mean; default value is 0.

      Type: DOUBLE
    - **stddev**
      The standard deviation; default value is 1.

      Type: DOUBLE
  - Returns

BOOLEAN TRUE, if successful.

Details

This procedure uses drand48_r.

Examples

```
CALL nzm..normal('A', 10, 10, 35.5, 48.7);
CALL nzm..list_matrices();
CALL nzm..delete_matrix('A');
```

```
 NORMAL
--------
 t
(1 row)
 LIST_MATRICES
---------------
 A
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

NORMAL - Matrix of Random, Normally Distributed Values - Simplified Version

This procedure creates a new matrix filled with normally distributed random values using drand48_r.

Usage

The NORMAL stored procedure has the following syntax:

- **NORMAL(matrixOut, numberOfRows, numberOfColumns)**
  - Parameters
    - **matrixOut**
      The name of the matrix to be generated.

Type: NVARCHAR(ANY)

- **numberOfRows**
  The number of rows to be included in the created matrix.

  Type: INT4

- **numberOfColumns**
  The number of columns to be included in the created matrix.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Details

This procedure uses drand48_r.

Examples

```
CALL nzm..normal('A', 10, 10);
CALL nzm..list_matrices();
CALL nzm..delete_matrix('A');
```

```
 NORMAL
--------
 t
(1 row)
 LIST_MATRICES
---------------
 A
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

POWER_ELEMENTS - Elementwise POWER Function

This procedure implements an elementwise raising of the specified block of elements to a power.

Usage

The POWER_ELEMENTS stored procedure has the following syntax:

- **POWER_ELEMENTS('matrixIn','matrixOut',power, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **power**
      The power to use.

      Type: DOUBLE
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..power_elements('A', 'B', 3 , 2, 2, 3, 3);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
```

```
CALL nzm..delete_matrix('B');

 SHAPE
-------
 t
(1 row)
 POWER_ELEMENTS
----------------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                     PRINT
-----------------------------------------------
 -- matrix: B --
1, 2, 3
4, 125, 0
6, 343, 512
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
```

```
(1 row)
```

Related Functions

- category matrix operations

POWER_ELEMENTS - Elementwise POWER Function (entire matrix operation)

This procedure implements an elementwise raising of elements to a power.

Usage

The POWER_ELEMENTS stored procedure has the following syntax:

- **POWER_ELEMENTS('matrixIn','matrixOut',power)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **power**
      The power to use.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..power_elements('A', 'B', 3);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
```

```
(1 row)
 POWER_ELEMENTS
----------------
 t
(1 row)
                   PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                        PRINT
---------------------------------------------------
 -- matrix: B --
1, 8, 27
64, 125, 0
216, 343, 512
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

PRINT - Print a Matrix

This procedure generates formatted print of a given matrix.

Usage

The PRINT stored procedure has the following syntax:

- **PRINT('matrixIn',r_style)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **r_style**
      A Boolean TRUE/FALSE value.

      Type: boolean
  - Returns
    NVARCHAR(16000) The matrix as a string

Examples

```
CALL nzm..CREATE_IDENTITY_MATRIX('A',4);
CALL nzm..PRINT('A', false);
CALL nzm..DELETE_MATRIX('A');
```

```
 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
                            PRINT
-------------------------------------------------------------
 -- matrix: A --
1, 0, 0, 0
0, 1, 0, 0
0, 0, 1, 0
0, 0, 0, 1
(1 row)
 DELETE_MATRIX
---------------
```

```
 t
(1 row)

CALL nzm..CREATE_IDENTITY_MATRIX('A',4);
CALL nzm..PRINT('A', true);
CALL nzm..DELETE_MATRIX('A');

 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
                               PRINT
---------------------------------------------------------
-----------
 -- matrix: A --
A<- matrix(c(1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1),4,4)
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

PRINT - Print a Matrix - Simplified Version

This procedure generates formatted print of a given matrix.

Usage

The PRINT stored procedure has the following syntax:

- **PRINT('matrixIn')**
  - Parameters

- **matrixIn**
  The name of the input matrix.

  Type: NVARCHAR(ANY)

- Returns
  NVARCHAR(16000) The matrix as a string

Examples

```
CALL nzm..CREATE_IDENTITY_MATRIX('A',4);
CALL nzm..PRINT('A');
CALL nzm..DELETE_MATRIX('A');
```

```
 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
                              PRINT
-------------------------------------------------------------
 -- matrix: A --
1, 0, 0, 0
0, 1, 0, 0
0, 0, 1, 0
0, 0, 0, 1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RADIANS_ELEMENTS - Elementwise RADIANS Function

This procedure implements an elementwise degrees to radians conversion.

Usage

The RADIANS_ELEMENTS stored procedure has the following syntax:

- **RADIANS_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('0,45,90,180,270,360',4,4,'A');
CALL nzm..radians_elements('A', 'B', 2, 2, 4, 4);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');


 SHAPE
-------
```

```
 t
(1 row)
 RADIANS_ELEMENTS
------------------
 t
(1 row)
                                      PRINT
---------------------------------------------------------------
------------------
 -- matrix: A --
0, 45, 90, 180
270, 360, 0, 45
90, 180, 270, 360
0, 45, 90, 180
(1 row)

PRINT
---------------------------------------------------------------
---------------------------------------------------------------
--------------------------------------------------------
 -- matrix: B --
0, 45, 90, 180
270, 6.2831853071796, 0, 0.785398163397 45
90, 3.1415926535898, 4.7123889803847, 6.2831853071796
0, 0.785398163397 45, 1.5707963267949, 3.1415926535898
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RADIANS_ELEMENTS - Elementwise RADIANS Function (entire matrix operation)

This procedure implements an elementwise degrees to radians degrees conversion.

Usage

The RADIANS_ELEMENTS stored procedure has the following syntax:

- **RADIANS_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('0,45,90,180,270,360',4,4,'A');
CALL nzm..radians_elements('A', 'B');
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 RADIANS_ELEMENTS
```

```
------------------
 t
(1 row)
                                      PRINT
----------------------------------------------------------------
------------------
 -- matrix: A --
0, 45, 90, 180
270, 360, 0, 45
90, 180, 270, 360
0, 45, 90, 180
(1 row)

PRINT
----------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------------------
 -- matrix: B --
0, 0.78539816339745, 1.5707963267949, 3.1415926535898
4.7123889803847, 6.2831853071796, 0, 0.78539816339745
1.5707963267949, 3.1415926535898, 4.7123889803847,
6.2831853071796
0, 0.78539816339745, 1.5707963267949, 3.1415926535898
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RCV2SIMPLE - Transforms a row/column/value table to a "Simple" Matrix Table

This procedure transforms a table in Row/Column/Value format to the "simple" matrix table. Input can be in the form of a table or a matrix.

Usage

The RCV2SIMPLE stored procedure has the following syntax:

- **RCV2SIMPLE**
  - Parameters
    - **paramString**
      The input parameters specification.

      Type: TEXT
    - **intable**
      This parameter is used when the input is in table form.

      Type: NVARCHAR(ANY)
    - **inmatrix**
      This parameter is used when the input is in matrix form.

      Type: NVARCHAR(ANY)
    - **inmeta**
      The name of the input metadata table created by SIMPLE2RCV_ADV.

      Type: NVARCHAR(ANY)
    - **outtable**
      The name of the output data table in simple format.

      Type: NVARCHAR(ANY)
  - Returns
    INTEGER The number of rows in the output table.

Details

A "simple" matrix table is a database table where each table row contains a row index value and the matrix element values of the corresponding matrix row. This procedure supports nominal attribute value composition, transforming 0/1 dummy variables back to the original values recorded in the dictionary tables listed in the metadata table. The number of matrix columns must be less than 1600. Input can be in the form of a table or a matrix.

Examples

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE, V3
DOUBLE);
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1, incolumnlist=., id=ID');
CALL NZM..RCV2SIMPLE('intable=RCV1,  inmeta=RCV_META1,
outtable=SIMPLE2');
SELECT * FROM SIMPLE2;
SELECT * FROM RCV1;
SELECT * FROM RCV_META1;
DROP TABLE SIMPLE1;
DROP TABLE SIMPLE2;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
              3
(1 row)
 RCV2SIMPLE
------------
          3
(1 row)
 ID |   V1    |   V2    |   V3
----+--------+--------+--------
  2 | 200001 | 200002 | 200003
  1 | 100001 | 100002 | 100003
  3 | 300001 | 300002 | 300003
(3 rows)
 ROW | COL | VALUE
-----+-----+--------
```

```
   1 |    1 | 100001
   1 |    2 | 100002
   1 |    3 | 100003
   3 |    1 | 300001
   3 |    2 | 300002
   3 |    3 | 300003
   2 |    1 | 200001
   2 |    2 | 200002
   2 |    3 | 200003
(9 rows)
 COLID | COLNAME | COLDICT | OUTCOLBEG | OUTCOLEND
-------+---------+---------+-----------+-----------
     2 | V2      |         |         2 |         2
     1 | V1      |         |         1 |         1
     3 | V3      |         |         3 |         3
(3 rows)
```

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE);
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1,
outmeta=RCV_META1, intable=SIMPLE1, incolumnlist=.,
id=ID');
CALL NZM..CREATE_MATRIX_FROM_TABLE('RCV1', 'MATRIX1', 3,
3);
-- Input the matrix name, rather than the table name
CALL NZM..RCV2SIMPLE('inmatrix=MATRIX1,
inmeta=RCV_META1, outtable=SIMPLE2');
SELECT * FROM SIMPLE2;
SELECT * FROM RCV1;
SELECT * FROM RCV_META1;
```

```
DROP TABLE SIMPLE1;
DROP TABLE SIMPLE2;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
CALL nzm..DELETE_MATRIX('MATRIX1');
```

```
  SIMPLE2RCV_ADV
----------------
              3
(1 row)
 CREATE_MATRIX_FROM_TABLE
--------------------------
 t
(1 row)
 RCV2SIMPLE
------------
          3
(1 row)
 ID |   V1   |   V2   |   V3
----+--------+--------+--------
  2 | 200001 | 200002 | 200003
  3 | 300001 | 300002 | 300003
  1 | 100001 | 100002 | 100003
(3 rows)
 ROW | COL | VALUE
-----+-----+--------
   2 |   1 | 200001
   2 |   2 | 200002
   2 |   3 | 200003
   1 |   1 | 100001
   1 |   2 | 100002
   1 |   3 | 100003
   3 |   1 | 300001
```

```
   3 |    2 | 300002
   3 |    3 | 300003
(9 rows)
 COLID | COLNAME | COLDICT | OUTCOLBEG | OUTCOLEND
-------+---------+---------+-----------+-----------
     1 | V1      |         |         1 |         1
     3 | V3      |         |         3 |         3
     2 | V2      |         |         2 |         2
(3 rows)
 DELETE_MATRIX
---------------
 t
(1 row)
```

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE);
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
-- Treat V1 and V3 as nominal attributes
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1,
outmeta=RCV_META1, intable=SIMPLE1, incolumnlist=.,
nomcolumnlist=V1;V3, id=ID');
CALL NZM..RCV2SIMPLE('intable=RCV1,  inmeta=RCV_META1,
outtable=SIMPLE2');
SELECT * FROM SIMPLE2;
SELECT * FROM RCV1;
SELECT COLID, COLNAME, OUTCOLBEG, OUTCOLEND FROM
RCV_META1;
DROP TABLE SIMPLE1;
DROP TABLE SIMPLE2;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
               3
(1 row)
 RCV2SIMPLE
------------
           3
(1 row)
 ID |   V1   |   V2   |   V3
----+--------+--------+--------
  1 | 100001 | 100002 | 100003
  3 | 300001 | 300002 | 300003
  2 | 200001 | 200002 | 200003
(3 rows)
 ROW | COL | VALUE
-----+-----+--------
   1 |   1 |      1
   1 |   2 |      0
   1 |   3 |      0
   1 |   4 | 100002
   1 |   5 |      1
   1 |   6 |      0
   1 |   7 |      0
   2 |   1 |      0
   2 |   2 |      1
   2 |   3 |      0
   2 |   4 | 200002
   2 |   5 |      0
   2 |   6 |      1
   2 |   7 |      0
   3 |   1 |      0
   3 |   2 |      0
```

```
   3 |    3 |       1
   3 |    4 | 300002
   3 |    5 |       0
   3 |    6 |       0
   3 |    7 |       1
(21 rows)
 COLID | COLNAME | OUTCOLBEG | OUTCOLEND
-------+---------+-----------+-----------
     2 | V2      |         4 |         4
     1 | V1      |         1 |         3
     3 | V3      |         5 |         7
(3 rows)
```

Related Functions

- category matrix operations

RCV2SIMPLE_NUM - Transforms a row/column/value Table to a "Simple" Matrix Table

Transforms a row/column/value table to a "simple" matrix table. Input can be in the form of a table or a matrix.

Usage

The RCV2SIMPLE_NUM stored procedure has the following syntax:

- **RCV2SIMPLE_NUM**
  - Parameters
    - **paramString**
      The input parameters specification.

      Type: TEXT
    - **intable**
      This parameter is used when the input is in table form.

      Type: NVARCHAR(ANY)
    - **colprefix**
      The prefix of the column names for the new table.

Type: NVARCHAR(ANY)

- **inmatrix**
  This parameter is used when the input is in matrix form.

  Type: NVARCHAR(ANY)

- **outtable**
  The name of the output data table in simple format.

  Type: NVARCHAR(ANY)

- Returns
  INTEGER The number of rows in the output table.

Details

A "simple" matrix table is a database table where each table row contains a row index value and the matrix element values of the corresponding matrix row. This procedure supports nominal attribute value composition, transforming 0/1 dummy variables back to the original values recorded in the dictionary tables listed in the metadata table. The number of matrix columns must be less than 1600. Input can be in the form of a table or a matrix.

Examples

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE, V3
DOUBLE);

INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);

INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);

INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);

CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1, incolumnlist=., id=ID');

CALL NZM..RCV2SIMPLE_NUM('intable=RCV1,  outtable=SIMPLE2');

SELECT * FROM SIMPLE2;

SELECT * FROM RCV1;

SELECT * FROM RCV_META1;

DROP TABLE SIMPLE1;

DROP TABLE SIMPLE2;

DROP TABLE RCV1;

DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
              3
(1 row)
```

```
 RCV2SIMPLE_NUM
----------------
              3
(1 row)
 ID |  COL1  |  COL2  |  COL3
----+--------+--------+--------
  2 | 200001 | 200002 | 200003
  3 | 300001 | 300002 | 300003
  1 | 100001 | 100002 | 100003
(3 rows)
 ROW | COL | VALUE
-----+-----+--------
   1 |   1 | 100001
   1 |   2 | 100002
   1 |   3 | 100003
   2 |   1 | 200001
   2 |   2 | 200002
   2 |   3 | 200003
   3 |   1 | 300001
   3 |   2 | 300002
   3 |   3 | 300003
(9 rows)
 COLID | COLNAME | COLDICT | OUTCOLBEG | OUTCOLEND
-------+---------+---------+-----------+-----------
     2 | V2      |         |         2 |         2
     1 | V1      |         |         1 |         1
     3 | V3      |         |         3 |         3
(3 rows)
```

CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE, V3 DOUBLE);

INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);

```
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1, incolumnlist=., id=ID');
CALL NZM..RCV2SIMPLE_NUM('intable=RCV1,  outtable=SIMPLE3,
colprefix=column');
SELECT * FROM SIMPLE3;
SELECT * FROM RCV1;
SELECT * FROM RCV_META1;
DROP TABLE SIMPLE1;
DROP TABLE SIMPLE3;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
  SIMPLE2RCV_ADV
----------------
              3
(1 row)
 RCV2SIMPLE_NUM
----------------
              3
(1 row)
 ID | COLUMN1 | COLUMN2 | COLUMN3
----+---------+---------+---------
  2 |  200001 |  200002 |  200003
  3 |  300001 |  300002 |  300003
  1 |  100001 |  100002 |  100003
(3 rows)
 ROW | COL | VALUE
-----+-----+--------
   1 |   1 | 100001
   1 |   2 | 100002
   1 |   3 | 100003
   2 |   1 | 200001
```

```
    2 |    2 | 200002
    2 |    3 | 200003
    3 |    1 | 300001
    3 |    2 | 300002
    3 |    3 | 300003
(9 rows)
 COLID | COLNAME | COLDICT | OUTCOLBEG | OUTCOLEND
-------+---------+---------+-----------+-----------
     1 | V1      |         |         1 |         1
     2 | V2      |         |         2 |         2
     3 | V3      |         |         3 |         3
(3 rows)
```

Related Functions

- category matrix operations

RED_MAX - Maximum Value of a Matrix

This procedure implements computation of the maximum value from a matrix reduction.

Usage

The RED_MAX stored procedure has the following syntax:

- **RED_MAX(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The maximum value in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT nzm..red_max('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
 RED_MAX
---------
        9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_MAX_ABS - Maximum Absolute Value of a Matrix

This procedure implements computation of the maximum absolute value from a matrix reduction.

Usage

The RED_MAX_ABS stored procedure has the following syntax:

- **RED_MAX_ABS(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The maximum absolute value in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT  nzm..red_max_abs('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
 RED_MAX_ABS
-------------
           9
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_MIN - Minimum Value of a Matrix

This procedure implements computation of the minimum value from a matrix reduction.

Usage

The RED_MIN stored procedure has the following syntax:

- **RED_MIN(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The minimum value in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT nzm..red_min('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
 RED_MIN
---------
      -2
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_MIN_ABS - Minimum Absolute Value of a Matrix

This procedure implements computation of the minimum absolute value from a matrix reduction.

Usage

The RED_MIN_ABS stored procedure has the following syntax:

- **RED_MIN_ABS(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The minimum absolute value in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT nzm..red_min_abs('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
```

```
-------
 t
(1 row)
 RED_MIN_ABS
-------------
           1
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_SSQ - Sum of Squares of Values of a Matrix

This procedure implements computation of the sum of squares of all values from a matrix reduction.

Usage

The RED_SSQ stored procedure has the following syntax:

- **RED_SSQ(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The sum of squares of values in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT nzm..red_ssq('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
 RED_SSQ
---------
      174
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_SUM - Sum Values of a Matrix

This procedure implements computation of the sum of all values from a matrix reduction.

Usage

The RED_SUM stored procedure has the following syntax:

- **RED_SUM(matrixName);**
  - Parameters
    - **matrixName**
      The name of the matrix.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The sum of values in the matrix.

Examples

```
CALL nzm..shape('6,-2,9, 4,1,6',2,3,'A');
SELECT nzm..red_sum('A');
CALL nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
 RED_SUM
---------
       24
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

RED_TRACE - Trace

This procedure implements calculation of a trace of the matrix.

Usage

The RED_TRACE stored procedure has the following syntax:

- **RED_TRACE('matrixIn')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.
      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The value of the trace.

Examples

```
CALL nzm..CREATE_IDENTITY_MATRIX('A',4);
CALL nzm..RED_TRACE('A');
CALL nzm..DELETE_MATRIX('A');
```

```
 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
 RED_TRACE
-----------
         4
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

REDUCE_TO_VECT - Reduce to vector

This stored procedure reduces specified database matrix objects to a vector object.

Usage

The REDUCE_TO_VECT function has the following syntax:

- **REDUCE_TO_VECT(NVARCHAR(ANY) inputMatrix, NVARCHAR(ANY) outputVector, NVARCHAR(ANY) expressionPrefix, NVARCHAR(ANY) expressionPostfix, NVARCHAR(ANY) orientation)**
  - Parameters
    - **inputMatrix**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **outputVector**
      The name of the resulting matrix.

      Type: NVARCHAR(ANY)
    - **expressionPrefix**
      The prefix of the aggregate expression that is used for the reduction. Typically, the prefix is the name of an aggregate function.

      Type: NVARCHAR(ANY)

- **expressionPostfix**
  The postfix of the aggregate expression that is used for the reduction. Typically, the postfix consists of parameters of an aggregate function.

  Type: NVARCHAR(ANY)

- **orientation**
  The orientation of the resulting vector object and the reduction operation. Values are 'r' for row and 'c' for column.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN true on success

Details

This stored procedure reduces specified database matrix objects to a vector object by using the specified aggregate expression. A vector object is a database matrix object with a single row or column. The following values of the orientation parameters determine the orientation of the vector object: - 'c' means that the aggregation is executed on rows of the input matrix and results in columns. - 'r' means that the aggregation is executed on columns of the input matrix and results in rows. The specified aggregate expression uses the following concatenation for the expressionPrefix argument and the expressionPostfix argument: expressionPrefix || "(" || matrix cell value || expressionPostfix || ")" For example, consider the pairs 'AVG','' or 'SQRT(VARIANCE', ')'.

Examples

```
call nzm..create_random_matrix('REDUCTION_TEST', 1000,
1000);
```

```
call
nzm..reduce_to_vect('REDUCTION_TEST','REDUCTION_TEST_c','
avg','','c');
```

```
call
nzm..reduce_to_vect('REDUCTION_TEST','REDUCTION_TEST_r','
SQRT(VARIANCE',')','r');
```

Related Functions

- category matrix operations

REDUCTION - Reductions MAX MIN SSQ SUM TRACE

This procedure implements ssq, min, max, and sum on all elements of the matrix.

Usage

The REDUCTION stored procedure has the following syntax:

- **REDUCTION('matrixIn','reduction_type')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **reduction_type**
      The reduction type. Must be one of the following: MAX MIN SSQ SUM TRACE.

      Type: NVARCHAR(ANY)
  - Returns
    DOUBLE The value of the calculation.

Examples

```
CALL nzm..CREATE_IDENTITY_MATRIX('A',4);
CALL nzm..REDUCTION('A', 'SUM');
CALL nzm..DELETE_MATRIX('A');
 CREATE_IDENTITY_MATRIX
------------------------
 t
(1 row)
 REDUCTION
-----------
         4
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

REMOVE - Remove Operation

This procedure implements the remove operation.

Usage

The REMOVE stored procedure has the following syntax:

- **REMOVE(NVARCHAR(ANY) matrixAname, NVARCHAR(ANY) matrixBname, NVARCHAR(ANY) matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixBname**
      The name of input matrix B, containing the indexes of elements to be removed from Matrix A.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C, a row vector of matrix A with removed elements.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

If Matrix A and B, specified by the parameters, are considered row vectors, the second matrix indicates which elements of the first matrix are to be removed. The output is a row vector. Matrix C must not exist prior to the operation.

Examples

```
CALL nzm..shape('21,32,13,34,55,56,27,68,79',3,3,'A');
CALL nzm..shape('1,15,5,7',2,2,'B');
CALL nzm..remove('A','B','C');
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..print('C');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
CALL nzm..delete_matrix('C');
 SHAPE
-------
```

```
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 REMOVE
--------
 t
(1 row)
                      PRINT
--------------------------------------------------
 -- matrix: A --
21, 32, 13
34, 55, 56
27, 68, 79
(1 row)
           PRINT
----------------------------
 -- matrix: B --
1, 15
5, 7
(1 row)
                 PRINT
----------------------------------------
 -- matrix: C --
32, 13, 34, 56, 68, 79
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

REPEAT - Matrix Repeat

This procedure creates a new matrix of repeated values.

Usage

The REPEAT stored procedure has the following syntax:

- **REPEAT('matrixIn','matrixOut',nrow,ncol)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **nrow**
      The row multiplier.

      Type: INT4
    - **ncol**
      The column multiplier.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Details

The new matrix is of size: nrow*matrixIn.rows x ncol*matrixIn.cols.

Examples

```
CALL nzm..SHAPE('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..REPEAT('A', 'B', 2, 2);
CALL nzm..PRINT('A' );
CALL nzm..PRINT('B' );
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 REPEAT
--------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                                                  PRINT
---------------------------------------------------------------
--------------------------------------------------------
 -- matrix: B --
1, 2, 3, 1, 2, 3
4, 5, 0, 4, 5, 0
6, 7, 8, 6, 7, 8
1, 2, 3, 1, 2, 3
4, 5, 0, 4, 5, 0
6, 7, 8, 6, 7, 8
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ROUND_ELEMENTS - Elementwise ROUND Function

This procedure implements An elementwise modulo operation for the specified block of elements

Usage

The ROUND_ELEMENTS stored procedure has the following syntax:

- **ROUND_ELEMENTS('matrixIn','matrixOut',precision, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **precision**
      The desired decimal precision.

      Type: INT4
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**

The first column of the input matrix to use.

Type: INT4

- **row_stop**
  The last row of the input matrix to use.

  Type: INT4

- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1.1111,2.222,3.3333,4.4444,5.5555,6.6666,7.7777,8.88
88,9.9999',3,3,'A');

CALL nzm..ROUND_ELEMENTS('A', 'B', 3 , 2, 2, 3, 3);

CALL nzm..PRINT('A' );

CALL nzm..PRINT('B' );

CALL nzm..DELETE_MATRIX('A' );

CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 ROUND_ELEMENTS
----------------
 t
(1 row)
                                        PRINT
---------------------------------------------------------------
----------------------
 -- matrix: A --
1.1111, 2.222, 3.3333
4.4444, 5.5555, 6.6666
7.7777, 8.8888, 9.9999
```

```
(1 row)
                                         PRINT
-----------------------------------------------------------------
------------------------
 -- matrix: B --
1.1111, 2.222, 3.3333
4.4444, 5.556, 6.667
7.7777, 8.889, 10
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

ROUND_ELEMENTS - Elementwise ROUND Function (entire matrix operation)

This procedure implements an elementwise modulo operation for the specified block of elements

Usage

The ROUND_ELEMENTS stored procedure has the following syntax:

- **ROUND_ELEMENTS('matrixIn','matrixOut',precision)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)

- **matrixOut**
  The name of the output matrix.

  Type: NVARCHAR(ANY)
- **precision**
  The desired decimal precision.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL
nzm..SHAPE('1.1111,2.222,3.3333,4.4444,5.5555,6.6666,7.7777,8.88
88,9.9999',3,3,'A');
CALL nzm..ROUND_ELEMENTS('A', 'B', 3 );
CALL nzm..PRINT('A' );
CALL nzm..PRINT('B' );
CALL nzm..DELETE_MATRIX('A' );
CALL nzm..DELETE_MATRIX('B' );
```

```
 SHAPE
-------
 t
(1 row)
 ROUND_ELEMENTS
----------------
 t
(1 row)
                                        PRINT
---------------------------------------------------------------
----------------------
 -- matrix: A --
1.1111, 2.222, 3.3333
4.4444, 5.5555, 6.6666
7.7777, 8.8888, 9.9999
(1 row)
                                  PRINT
```

```
------------------------------------------------------------
------------------
 -- matrix: B --
1.111, 2.222, 3.333
4.444, 5.556, 6.667
7.778, 8.889, 10
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SCALAR_OPERATION - Elementwise Scalar Operation

This procedure implements elementwise scalar operations on a specified block of elements.

Usage

The SCALAR_OPERATION stored procedure has the following syntax:

- **SCALAR_OPERATION('matrixIn','matrixOut','operator',value, row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)

- **operator**
  The operator to use. Must be one of the following: + - * / % ^ & |

  Type: NVARCHAR(ANY)
- **value**
  The value.

  Type: DOUBLE
- **row_start**
  The first row of the input matrix to use.

  Type: INT4
- **col_start**
  The first column of the input matrix to use.

  Type: INT4
- **row_stop**
  The last row of the input matrix to use.

  Type: INT4
- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..scalar_operation('A', 'B', '+', 4, 2,2,3,3);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 SCALAR_OPERATION
------------------
 t
```

```
(1 row)
                    PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                     PRINT
-------------------------------------------
 -- matrix: B --
1, 2, 3
4, 9, 4
6, 11, 12
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SCALAR_OPERATION - Elementwise Scalar Operation (entire matrix operation)

This procedure implements elementwise scalar operations.

Usage

The SCALAR_OPERATION stored procedure has the following syntax:

- **SCALAR_OPERATION('matrixIn','matrixOut','operator',value)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.
      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.
      Type: NVARCHAR(ANY)
    - **operator**
      The operator to use. Must be one of the following: + - * / % ^ & |
      Type: NVARCHAR(ANY)
    - **value**
      The value.
      Type: DOUBLE
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..scalar_operation('A', 'B', '*', 2.2);
CALL nzm..print('A');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 SCALAR_OPERATION
------------------
 t
(1 row)
```

```
                     PRINT
------------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
(1 row)
                              PRINT
-----------------------------------------------------------
--
 -- matrix: B --
2.2, 4.4, 6.6
8.8, 11, 0
13.2, 15.4, 17.6
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SCALE - Scale the Elements of a Matrix

This procedure computes C = A f, where A and C are matrices and f is a real number.

Usage

The SCALE stored procedure has the following syntax:

- **SCALE(NVARCHAR(ANY) matrixA, DOUBLE factor, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **factor**
      The multiplication factor.

      Type: DOUBLE
    - **matrixC**
      The name of matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
call nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
call nzm..scale('A', '5', 'B');
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 SCALE
-------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: A --
1, 2, 3
4, 5, 0
6, 7, 8
```

```
(1 row)
                         PRINT
----------------------------------------------------------
 -- matrix: B --
5, 10, 15
20, 25, 0
30, 35, 40
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SET_BLOCK_SIZE - Set the Block Size for the Data Distribution

This procedure sets the block size for the 2-D block-cyclic data distribution.

Usage

The SET_BLOCK_SIZE stored procedure has the following syntax:

- **SET_BLOCK_SIZE(INT4 blockSizeRow, INT4 blockSizeCol);**
  - Parameters
    - **blockSizeRow**
      The row block size.

      Type: INT4
    - **blockSizeCol**
      The column block size.

Type: INT4

- Returns
  BOOLEAN TRUE always.

Details

This procedure is rarely needed, since nzm..initialize() typically sets the block size to a reasonable default value.

Examples

```
call nzm..set_block_size(2,2);
 SET_BLOCK_SIZE
----------------
 t
(1 row)
```

Related Functions

- category matrix operations

SET_GRID_SIZE - Set the Process Grid Size for the Matrix Engine.

This procedure sets the process grid size for the parallel matrix engine.

Usage

The SET_GRID_SIZE stored procedure has the following syntax:

- **SET_GRID_SIZE(INT4 gridSizeRow, INT4 gridSizeCol);**
  - Parameters
    - **gridSizeRow**
      The number of rows in the process grid.

      Type: INT4
    - **gridSizeCol**
      The number of columns in the process grid.

      Type: INT4
  - Returns
    BOOLEAN TRUE always.

Details

This procedure is rarely needed, since nzm..initialize() typically sets the process grid size to a reasonable default value. To resize the process grid with a redistribution of currently-existing matrices use the nzm..SET_GRID_SIZE_WITH_REDISTRIBUTE procedure.

Examples

```
call nzm..set_grid_size(2,2);
  SET_GRID_SIZE
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SET_GRID_SIZE_WITH_REDISTRIBUTE - Set the Process Grid Size for the Matrix Engine with Redistribution

This procedure sets the process grid size for the parallel matrix engine with redistribution of all currently existing matrices.

Usage

The SET_GRID_SIZE_WITH_REDISTRIBUTE stored procedure has the following syntax:

- **SET_GRID_SIZE_WITH_REDISTRIBUTE(INT4 gridSizeRow, INT4 gridSizeCol);**
  - Parameters
    - **gridSizeRow**
      The number of rows in the process grid.

      Type: INT4
    - **gridSizeCol**
      The number of columns in the process grid.

      Type: INT4
  - Returns
    BOOLEAN TRUE always.

Details

This procedure is rarely needed, since nzm..initialize() typically sets the process grid size to a reasonable default value.

Related Functions

- category matrix operations

SET_VALUE - Set the Value of a Matrix Element

This procedure sets the value of the specified matrix element.

Usage

The SET_VALUE stored procedure has the following syntax:

- **SET_VALUE(NVARCHAR(ANY) mat_name, INT4 inrow, INT4 incol);**
  - Parameters
    - **mat_name**
      The name of the matrix.

      Type: NVARCHAR(ANY)
    - **inrow**
      The row index of the element.

      Type: INT4
    - **incol**
      The column index of the element.

      Type: INT4
    - **inval**
      The value for the matrix element.

      Type: DOUBLE
  - Returns
    BOOLEAN TRUE always.

Details

This procedure is more efficient when the number of values is relatively small. For setting large numbers of values, use alternate approaches that process data in bulk.

Examples

```
call nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
call nzm..set_value('A', 1, 1, 878);
call nzm..print('A');
call nzm..delete_matrix('A');
 SHAPE
-------
 t
(1 row)
 SET_VALUE
-----------
```

```
 t
(1 row)
                          PRINT
-------------------------------------------
 -- matrix: A --
878, 2, 3
4, 5, 0
6, 7, 8
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SHAPE - Cyclically Fill a Matrix with Elements from a List

This procedure creates a matrix filled cyclically with elements from a list.

Usage

The SHAPE stored procedure has the following syntax:

- **SHAPE(valuelist, rows, cols, matrixCname);**
  - Parameters
    - **valuelist**
      A comma-separated list of doubles.

      Type: NVARCHAR(ANY)
    - **rows**
      The number of rows.

      Type: INT4
    - **cols**
      The number of columns.

      Type: INT4

- **matrixCname**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE, if successful.

Details

This procedure creates a matrix that is filled cyclically based on the values in the valuelist parameter. For example, if a matrix of size 3 x 3 is created with a list of "2,3,5,7" the result is 2 3 5 | 7 2 3 | 5 7 2. Note that the well-formedness of the list is not tested.

Examples

```
call nzm..shape('2,3,5,7',3,3,'A');
call nzm..print('A');
call nzm..delete_matrix('A');
```

```
 SHAPE
-------
 t
(1 row)
                  PRINT
-----------------------------------------
 -- matrix: A --
2, 3, 5
7, 2, 3
5, 7, 2
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SHAPEMTX - Cyclically Fill a Matrix with Elements from a Row Vector

This procedure creates a matrix cyclically filled with elements from a row vector.

Usage

The SHAPEMTX stored procedure has the following syntax:

- **SHAPEMTX(matrixAname, rows, cols, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of a one-row matrix.

      Type: NVARCHAR(ANY)
    - **rows**
      The number of rows.

      Type: INT4
    - **cols**
      The number of columns.

      Type: INT4
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Details

The row vector is in the form of a one-row matrix. For numerical values passed as a string, see SHAPE.

Examples

```
call nzm..shape('1,2,3,4',1,4,'A');
call nzm..shapeMtx('A', 3, 2, 'B');
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');
```

SHAPE

```
-------
 t
(1 row)
 SHAPEMTX
----------
 t
(1 row)
           PRINT
----------------------------
 -- matrix: A --
1, 2, 3, 4
(1 row)
             PRINT
--------------------------------
 -- matrix: B --
1, 2
3, 4
1, 2
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SIGN_REVERSE - Elementwise Sign Reversal

This procedure implements a sign reversal on the specified block of elements.

Usage

The SIGN_REVERSE stored procedure has the following syntax:

- **SIGN_REVERSE('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **col_start**
      The first column of the input matrix to use.

      Type: INT4
    - **row_stop**
      The last row of the input matrix to use.

      Type: INT4
    - **col_stop**
      The last column of the input matrix to use.

      Type: INT4
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
call nzm..shape('1,2,3,4,5,0,6,7,8,9',4,4,'A');
call nzm..sign_reverse('A', 'B', 2, 2, 3, 3);
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');


 SHAPE
-------
```

```
 t
(1 row)
 SIGN_REVERSE
--------------
 t
(1 row)
                             PRINT
--------------------------------------------------------------
 -- matrix: A --
1, 2, 3, 4
5, 0, 6, 7
8, 9, 1, 2
3, 4, 5, 0
(1 row)
                               PRINT
----------------------------------------------------------------
--
 -- matrix: B --
1, 2, 3, 4
5, -0, -6, 7
8, -9, -1, 2
3, 4, 5, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SIGN_REVERSE - Elementwise Sign Reversal (entire matrix operation)

This procedure implements a sign reversal on the specified block of elements.

Usage

The SIGN_REVERSE stored procedure has the following syntax:

- **SIGN_REVERSE('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
call nzm..shape('1,2,3,4,5,0,6,7,8,9',4,4,'A');
call nzm..sign_reverse('A', 'B');
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');
 SHAPE
-------
 t
(1 row)
 SIGN_REVERSE
--------------
 t
(1 row)
```

```
                        PRINT
-----------------------------------------------------------
 -- matrix: A --
1, 2, 3, 4
5, 0, 6, 7
8, 9, 1, 2
3, 4, 5, 0
(1 row)
                                PRINT
-------------------------------------------------------------
--------------
 -- matrix: B --
-1, -2, -3, -4
-5, -0, -6, -7
-8, -9, -1, -2
-3, -4, -5, -0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SIMPLE2RCV - Transforms a "Simple" Matrix Table to row/column/value Representation

This procedure transforms a "simple" matrix table to a row/column/value representation.

Usage

The SIMPLE2RCV stored procedure has the following syntax:

- **SIMPLE2RCV('inTable', 'idColumn', 'colPropertiesTable','outTable','byValue')**
  - Parameters
    - **inTable**
      The name of the input table.

      Type: NVARCHAR(ANY)
    - **idColumn**
      The name of the column containing the row index values.

      Type: NVARCHAR(ANY)
    - **colPropertiesTable**
      the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPER-TIES().

      If the parameter is undefined, the input table column properties will be detected automatically.

      Type: NVARCHAR(ANY)
    - **outTable**
      The name of the output table.

      Type: NVARCHAR(ANY)
    - **byValue**
      The name of column which will be used to distribute data. In the result table it is called as id_task.

      Type: NVARCHAR(ANY)
  - Returns
    INTEGER The number of matrix columns found in the input table.

Details

The input table must have a column containing the row indices. The remaining columns, named "v1", "v2", and so on, contain the matrix element values. The number of matrix columns must be less than 1600. To add a consecutive ROW index column to a table containing a non-consecutive unique ID column, a query such as: CREATE TABLE SIMPLE3 AS SELECT *, ROW_NUMBER() OVER (ORDER BY ID) AS ROW FROM SIMPLE2; can be used. The nzm..CREATE_MATRIX_FROM_TABLE procedure can be used to create an nzMatrix from the row/column/value table produced by this procedure.

Related Functions

- category matrix operations

SIMPLE2RCV_ADV - Transforms a Table to row/column/value Representation

This procedure transforms a table to row/column/value representation.

Usage

The SIMPLE2RCV_ADV stored procedure has the following syntax:

- **SIMPLE2RCV_ADV(NVARCHAR(ANY) paramString)**
  - Parameters
    - **paramString**
      The input parameters specification.

      Type: NVARCHAR(ANY)
    - **outtable**
      The name of the output row/column/value table.

      Type: NVARCHAR(ANY)
    - **outmeta**
      The name of the output metadata table.

      Type: NVARCHAR(ANY)
    - **intable**
      The name of the input table.

      Type: NVARCHAR(ANY)
    - **id**
      The name of the column containing unique ID values.

      Type: NVARCHAR(ANY)
    - **incolumnlist**
      The list of names of the input columns. Column names are separated by semicolons. A dot matches all columns. A dash followed by a column name excludes the named column.

      Type: NVARCHAR(ANY)
    - **nomcolumnlist**
      The list of names of the input columns representing nominal attributes to be decomposed.

      Type: NVARCHAR(ANY)
    - **colPropertiesTable**
      the name of the table where the column properties definitions are stored

      Type: NVARCHAR(ANY)

      Default: ''
  - Returns
    INTEGER The number of input table attribute columns used.

Details

A metadata table is produced to record the mapping of input columns to output columns. Nominal attributes are supported. Dictionary tables referenced in the metadata table list the unique values for each nominal attribute. Decomposition of nominal factor values into separate columns is also supported.

Examples

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE);
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
-- Use columns V1, V2, and V3
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1,
outmeta=RCV_META1, intable=SIMPLE1,
incolumnlist=V1;V2;V3, id=ID');
SELECT 'SIMPLE1',* FROM  SIMPLE1 ORDER BY 1,2,3;
SELECT 'RCV_META1',* FROM  RCV_META1 ORDER BY 1,2,3;
SELECT 'RCV1',* FROM  RCV1 ORDER BY 1,2,3;
DROP TABLE SIMPLE1;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
              3
(1 row)
 ?COLUMN? | ID |   V1   |   V2   |   V3
----------+----+--------+--------+--------
 SIMPLE1  |  1 | 100001 | 100002 | 100003
 SIMPLE1  |  4 | 200001 | 200002 | 200003
 SIMPLE1  |  9 | 300001 | 300002 | 300003
(3 rows)
 ?COLUMN?  | COLID | COLNAME | COLDICT | OUTCOLBEG |
OUTCOLEND
-----------+-------+---------+---------+-----------
```

```
+-----------
 RCV_META1 |      1 | V1       |          |         1 |          1
 RCV_META1 |      2 | V2       |          |         2 |          2
 RCV_META1 |      3 | V3       |          |         3 |          3
(3 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV1     |   1 |   1 | 100001
 RCV1     |   1 |   2 | 100002
 RCV1     |   1 |   3 | 100003
 RCV1     |   2 |   1 | 200001
 RCV1     |   2 |   2 | 200002
 RCV1     |   2 |   3 | 200003
 RCV1     |   3 |   1 | 300001
 RCV1     |   3 |   2 | 300002
 RCV1     |   3 |   3 | 300003
(9 rows)
```

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE, V3
DOUBLE);
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003);
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003);
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003);
-- Use all columns except V2
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1, incolumnlist=.;-V2, id=ID');
SELECT 'SIMPLE1',* FROM  SIMPLE1 ORDER BY 1,2,3;
SELECT 'RCV_META1',* FROM  RCV_META1 ORDER BY 1,2,3;
SELECT 'RCV1',* FROM  RCV1 ORDER BY 1,2,3;
DROP TABLE SIMPLE1;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
```

```
              2
(1 row)
 ?COLUMN? | ID |   V1   |   V2   |   V3
----------+----+--------+--------+--------
 SIMPLE1  |  1 | 100001 | 100002 | 100003
 SIMPLE1  |  4 | 200001 | 200002 | 200003
 SIMPLE1  |  9 | 300001 | 300002 | 300003
(3 rows)
 ?COLUMN?  | COLID | COLNAME | COLDICT | OUTCOLBEG |
OUTCOLEND
-----------+-------+---------+---------+-----------
+-----------
 RCV_META1 |     1 | V1      |         |         1 |
1
 RCV_META1 |     2 | V3      |         |         2 |
2
(2 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV1     |   1 |   1 | 100001
 RCV1     |   1 |   2 | 100003
 RCV1     |   2 |   1 | 200001
 RCV1     |   2 |   2 | 200003
 RCV1     |   3 |   1 | 300001
 RCV1     |   3 |   2 | 300003
(6 rows)
```

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE, N1 VARCHAR(5), N2 NVARCHAR(5));
INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003,
'one','jeden');
INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003,
'two','dwa');
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003,
```

```
'three','trzy');
-- Treat N1 and N2 as nominal attributes
CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1, incolumnlist=., nomcolumnlist=N1;N2, id=ID');
SELECT 'SIMPLE1',* FROM  SIMPLE1 ORDER BY 1,2,3;
SELECT 'RCV_META1', COLID, COLNAME, OUTCOLBEG, OUTCOLEND  FROM
RCV_META1 ORDER BY 1,2,3;
SELECT 'RCV1',* FROM  RCV1 ORDER BY 1,2,3;
DROP TABLE SIMPLE1;
DROP TABLE RCV1;
DROP TABLE RCV_META1;
```

```
 SIMPLE2RCV_ADV
----------------
              5
(1 row)
 ?COLUMN? | ID |   V1   |   V2   |   V3   |  N1   |  N2
----------+----+--------+--------+--------+-------+-------
 SIMPLE1  |  1 | 100001 | 100002 | 100003 | one   | jeden
 SIMPLE1  |  4 | 200001 | 200002 | 200003 | two   | dwa
 SIMPLE1  |  9 | 300001 | 300002 | 300003 | three | trzy
(3 rows)
 ?COLUMN?  | COLID | COLNAME | OUTCOLBEG | OUTCOLEND
-----------+-------+---------+-----------+-----------
 RCV_META1 |     1 | V1      |         1 |         1
 RCV_META1 |     2 | V2      |         2 |         2
 RCV_META1 |     3 | V3      |         3 |         3
 RCV_META1 |     4 | N1      |         4 |         6
 RCV_META1 |     5 | N2      |         7 |         9
(5 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV1     |   1 |   1 | 100001
 RCV1     |   1 |   2 | 100002
 RCV1     |   1 |   3 | 100003
```

```
 RCV1      |    1 |    4 |       1
 RCV1      |    1 |    5 |       0
 RCV1      |    1 |    6 |       0
 RCV1      |    1 |    7 |       0
 RCV1      |    1 |    8 |       1
 RCV1      |    1 |    9 |       0
 RCV1      |    2 |    1 |  200001
 RCV1      |    2 |    2 |  200002
 RCV1      |    2 |    3 |  200003
 RCV1      |    2 |    4 |       0
 RCV1      |    2 |    5 |       0
 RCV1      |    2 |    6 |       1
 RCV1      |    2 |    7 |       1
 RCV1      |    2 |    8 |       0
 RCV1      |    2 |    9 |       0
 RCV1      |    3 |    1 |  300001
 RCV1      |    3 |    2 |  300002
 RCV1      |    3 |    3 |  300003
 RCV1      |    3 |    4 |       0
 RCV1      |    3 |    5 |       1
 RCV1      |    3 |    6 |       0
 RCV1      |    3 |    7 |       0
 RCV1      |    3 |    8 |       0
 RCV1      |    3 |    9 |       1
(27 rows)
```

```
CREATE TABLE SIMPLE1 (ID INTEGER, V1 DOUBLE, V2 DOUBLE,
V3 DOUBLE, N1 VARCHAR(5), N2 NVARCHAR(5));

INSERT INTO SIMPLE1 VALUES(1, 100001, 100002, 100003,
'one','jeden');

INSERT INTO SIMPLE1 VALUES(4, 200001, 200002, 200003,
'two','dwa');
```

```
INSERT INTO SIMPLE1 VALUES(9, 300001, 300002, 300003,
'three','trzy');

CALL nza..COLUMN_PROPERTIES('intable=SIMPLE1,outtable=CPT1');

CALL nza..SET_COLUMN_PROPERTIES('intable=SIMPLE1
,colPropertiesTable=CPT1,incolumn=ID:id');

-- Treat N1 and N2 as nominal attributes

CALL NZM..SIMPLE2RCV_ADV('outtable=RCV1, outmeta=RCV_META1,
intable=SIMPLE1,colPropertiesTable=CPT1');

SELECT 'SIMPLE1',* FROM  SIMPLE1 ORDER BY 1,2,3;

SELECT 'RCV_META1', COLID, COLNAME, OUTCOLBEG, OUTCOLEND  FROM
RCV_META1 ORDER BY 1,2,3;

SELECT 'CPT1',* FROM  CPT1 ORDER BY 1,2,3;

SELECT 'RCV1',* FROM  RCV1 ORDER BY 1,2,3;

DROP TABLE SIMPLE1;

DROP TABLE RCV1;

DROP TABLE RCV_META1;

DROP TABLE CPT1;
```

```
 COLUMN_PROPERTIES
-------------------
                 6
(1 row)
 SET_COLUMN_PROPERTIES
-----------------------
 t
(1 row)
 SIMPLE2RCV_ADV
----------------
              5
(1 row)
 ?COLUMN? | ID |   V1   |   V2   |   V3   |  N1   |  N2
----------+----+--------+--------+--------+-------+-------
 SIMPLE1  |  1 | 100001 | 100002 | 100003 | one   | jeden
 SIMPLE1  |  4 | 200001 | 200002 | 200003 | two   | dwa
 SIMPLE1  |  9 | 300001 | 300002 | 300003 | three | trzy
```

```
(3 rows)
 ?COLUMN?  | COLID | COLNAME | OUTCOLBEG | OUTCOLEND
-----------+-------+---------+-----------+-----------
 RCV_META1 |     1 | V1      |         1 |         1
 RCV_META1 |     2 | V2      |         2 |         2
 RCV_META1 |     3 | V3      |         3 |         3
 RCV_META1 |     4 | N1      |         4 |         6
 RCV_META1 |     5 | N2      |         7 |         9
(5 rows)
 ?COLUMN? | IDCOL | COLNAME |          COLDATATYPE
| COLTYPE | COLROLE | COLWEIGHT
----------+-------+---------
+--------------------------------+---------+---------
+-----------
 CPT1     |     1 | ID      | INTEGER
| cont    | id      |         1
 CPT1     |     2 | V1      | DOUBLE PRECISION
| cont    | input   |         1
 CPT1     |     3 | V2      | DOUBLE PRECISION
| cont    | input   |         1
 CPT1     |     4 | V3      | DOUBLE PRECISION
| cont    | input   |         1
 CPT1     |     5 | N1      | CHARACTER VARYING(5)
| nom     | input   |         1
 CPT1     |     6 | N2      | NATIONAL CHARACTER
VARYING(5) | nom     | input   |         1
(6 rows)
 ?COLUMN? | ROW | COL | VALUE
----------+-----+-----+--------
 RCV1     |   1 |   1 | 100001
 RCV1     |   1 |   2 | 100002
 RCV1     |   1 |   3 | 100003
 RCV1     |   1 |   4 |      1
 RCV1     |   1 |   5 |      0
 RCV1     |   1 |   6 |      0
```

```
RCV1      |   1 |   7 |      0
RCV1      |   1 |   8 |      1
RCV1      |   1 |   9 |      0
RCV1      |   2 |   1 | 200001
RCV1      |   2 |   2 | 200002
RCV1      |   2 |   3 | 200003
RCV1      |   2 |   4 |      0
RCV1      |   2 |   5 |      0
RCV1      |   2 |   6 |      1
RCV1      |   2 |   7 |      1
RCV1      |   2 |   8 |      0
RCV1      |   2 |   9 |      0
RCV1      |   3 |   1 | 300001
RCV1      |   3 |   2 | 300002
RCV1      |   3 |   3 | 300003
RCV1      |   3 |   4 |      0
RCV1      |   3 |   5 |      1
RCV1      |   3 |   6 |      0
RCV1      |   3 |   7 |      0
RCV1      |   3 |   8 |      0
RCV1      |   3 |   9 |      1
(27 rows)
```

Related Functions

- category matrix operations

SOLVE - Solve the Matrix Equation A X = B

This procedure solves the equation A X = B for X, where A, B, and X are matrices.

Usage

The SOLVE stored procedure has the following syntax:

- **SOLVE(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) matrixX);**

- Parameters
  - **matrixA**
    The name of input matrix A.

    Type: NVARCHAR(ANY)
  - **matrixB**
    The name of input matrix B.

    Type: NVARCHAR(ANY)
  - **matrixX**
    The name of output matrix X.

    Type: NVARCHAR(ANY)
- Returns
  BOOLEAN TRUE always.

Details

This procedure assumes that matrix A has full rank.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..shape('10,500,10,-900',3,3,'B');
CALL nzm..solve('A', 'B', 'X');
CALL nzm..gemm('A', 'X', 'B1');
CALL nzm..subtract('B', 'B1', 'B0');
CALL nzm..print('B0');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix  ('B');
CALL nzm..delete_matrix  ('B0');
CALL nzm..delete_matrix  ('B1');
CALL nzm..delete_matrix  ('X');
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
```

```
 t
(1 row)
 SOLVE
-------
 t
(1 row)
 GEMM
------
 t
(1 row)
 SUBTRACT
----------
 t
(1 row)

PRINT
------------------------------------------------------------------
------------------------------------------------------------------
---------------------------
 -- matrix: B0 --
-1.1368683772162e-13, 2.2737367544323e-13, 2.8421709430404e-14
-2.2737367544323e-13, 9.0949470177293e-13, 1.7053025658242e-13
0, 0, 0
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
```

```
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SOLVE_LINEAR_LEAST_SQUARES - Solve Linear Least Squares Problem

This procedure finds the linear least squares solution X to the matrix equation A X = B.

Usage

The SOLVE_LINEAR_LEAST_SQUARES stored procedure has the following syntax:

- **SOLVE_LINEAR_LEAST_SQUARES(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) matrixX);**
  - Parameters
    - **matrixA**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixB**
      The name of input matrix B.

      Type: NVARCHAR(ANY)
    - **matrixX**
      The name of output matrix X.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Details

This procedure assumes that matrix A has full rank.

Examples

```
CALL nzm..shape('1,2,3,4,5,0,6,7,8',3,3,'A');
CALL nzm..shape('10,500,10,-900',3,3,'B');
CALL nzm..solve_linear_least_squares('A', 'B', 'X');
CALL nzm..print('X');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix  ('B');
CALL nzm..delete_matrix  ('X');
```

```
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 SOLVE_LINEAR_LEAST_SQUARES
----------------------------
 t
(1 row)

PRINT
----------------------------------------------------------------
----------------------------------------------------------------
----------------------------------------------
 -- matrix: X --
141.66666666667, -1118.3333333333, -91.666666666666
-293.33333333333, 896.66666666667, 173.33333333333
151.66666666667, -58.333333333333, -81.666666666667
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SQRT_ELEMENTS - Elementwise SQRT

This procedure implements an elementwise square root calculation for the specified block of elements.

Usage

The SQRT_ELEMENTS stored procedure has the following syntax:

- **SQRT_ELEMENTS('matrixIn', 'matrixOut', row_start, col_start, row_stop, col_stop)**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **row_start**
      The first row of the input matrix to use.

      Type: INT4
    - **The**
      first column of the input matrix to use.

Type: INT4

- **row_stop**
  The last row of the input matrix to use.

  Type: INT4

- **col_stop**
  The last column of the input matrix to use.

  Type: INT4

- Returns
  BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('2,9,49,64,81',3,3,'A');
CALL nzm..SQRT_ELEMENTS('A', 'B', 2, 2, 3, 3);
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix  ('B');
```

```
 SHAPE
-------
 t
(1 row)
 SQRT_ELEMENTS
---------------
 t
(1 row)
                          PRINT
-----------------------------------------------------------
 -- matrix: B --
2, 9, 49
64, 9, 1.4142135623731
9, 7, 8
(1 row)
 DELETE_MATRIX
---------------
 t
```

```
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SQRT_ELEMENTS - Elementwise SQRT (entire matrix operation)

This procedure implements elementwise square root calculation

Usage

The SQRT_ELEMENTS stored procedure has the following syntax:

- **SQRT_ELEMENTS('matrixIn', 'matrixOut')**
  - Parameters
    - **matrixIn**
      The name of the input matrix.

      Type: NVARCHAR(ANY)
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE, if successful.

Examples

```
CALL nzm..shape('2,9,49,64,81',3,3,'A');
CALL nzm..SQRT_ELEMENTS('A', 'B');
CALL nzm..print('B');
CALL nzm..delete_matrix('A');
CALL nzm..delete_matrix  ('B');
 SHAPE
-------
 t
```

```
(1 row)
 SQRT_ELEMENTS
---------------
 t
(1 row)
                                PRINT
---------------------------------------------------------------
------
 -- matrix: B --
1.4142135623731, 3, 7
8, 9, 1.4142135623731
3, 7, 8
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SUBTRACT - Matrix Subtraction

This procedure computes C = A - B, where A, B, and C are matrices.

Usage

The SUBTRACT stored procedure has the following syntax:

- **SUBTRACT(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixB, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**

The name of input matrix A.

Type: NVARCHAR(ANY)

- **matrixB**
  The name of input matrix B.

  Type: NVARCHAR(ANY)

- **matrixC**
  The name of output matrix C.

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9,0',3,3,'A');
call nzm..shape('0,9,8,7,6,5,4,3,2,1',3,3,'B');
call nzm..subtract('A','B','C');
call nzm..print('C');
call nzm..delete_matrix('A');
call nzm..delete_matrix  ('B');
call nzm..delete_matrix  ('C');
 SHAPE
-------
 t
(1 row)
 SHAPE
-------
 t
(1 row)
 SUBTRACT
----------
 t
(1 row)
                    PRINT
---------------------------------------------
```

```
 -- matrix: C --
1, -7, -5
-3, -1, 1
3, 5, 7
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

SVD - Singular Value Decomposition

This procedure computes the Singular Value Decomposition A = U * SIGMA * transpose(V) of a matrix.

Usage

The SVD stored procedure has the following syntax:

- **SVD(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixU, NVARCHAR(ANY) matrixS, NVARCHAR(ANY) matrixVT);**
  - Parameters
    - **matrixA**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixU**
      The name of output matrix U.

Type: NVARCHAR(ANY)

- **matrixS**
  The name of the one-column output matrix S.

  Type: NVARCHAR(ANY)

- **matrixVT**
  The name of output matrix transpose(V).

  Type: NVARCHAR(ANY)

- Returns
  BOOLEAN TRUE always.

Details

Use "call nzm..vec_to_diag('S','SIGMA');" to create the diagonal matrix SIGMA from the one-column matrix S.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9,0',3,3,'A');
call nzm..svd('A', 'U', 'S', 'VT');
call nzm..vec_to_diag('S','SIGMA');
call nzm..gemm('U', 'SIGMA', 'USIGMA');
call nzm..gemm('USIGMA', 'VT', 'A1');
call nzm..subtract('A', 'A1', 'A0');
call nzm..print('A0');
call nzm..delete_matrix('A');
call nzm..delete_matrix  ('U');
call nzm..delete_matrix  ('S');
call nzm..delete_matrix  ('VT');
call nzm..delete_matrix  ('SIGMA');
call nzm..delete_matrix  ('USIGMA');
call nzm..delete_matrix  ('A0');
call nzm..delete_matrix  ('A1');
```

```
 SHAPE
-------
 t
```

```
(1 row)
 SVD
-----
 t
(1 row)
 VEC_TO_DIAG
-------------
 t
(1 row)
 GEMM
------
 t
(1 row)
 GEMM
------
 t
(1 row)
 SUBTRACT
----------
 t
(1 row)

PRINT
-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
----------------------
 -- matrix: A0 --
-1.7763568394003e-15, -6.2172489379009e-15, -3.1086244689504e-
15
-1.7763568394003e-15, -1.7763568394003e-15, -4.4408920985006e-
15
-2.6645352591004e-15, -7.105427357601e-15, -7.105427357601e-15
(1 row)
 DELETE_MATRIX
```

```
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

TRANSPOSE - Matrix Transpose

This procedure computes the transposed matrix C from matrix A.

Usage

The TRANSPOSE stored procedure has the following syntax:

- **TRANSPOSE(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
call nzm..shape('1,2,3,4,5,6,7,8,9,0',2,4,'A');
call nzm..transpose('A', 'B');
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');
 SHAPE
-------
 t
(1 row)
 TRANSPOSE
-----------
 t
(1 row)
```

```
                  PRINT
----------------------------------------
 -- matrix: A --
1, 2, 3, 4
5, 6, 7, 8
(1 row)
                 PRINT
--------------------------------------
 -- matrix: B --
1, 5
2, 6
3, 7
4, 8
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

UNIFORM - Matrix of Random, Uniformly Distributed Values.

This procedure creates a new matrix filled with uniformly distributed random values greater than or equal to zero and less than 1.

Usage

The UNIFORM stored procedure has the following syntax:

- **UINFORM(matrixOut, numberOfRows, numberOfColumns)**
  - Parameters
    - **matrixOut**
      The name of the output matrix.

      Type: NVARCHAR(ANY)
    - **numberOfRows**
      The number of rows to include in the new matrix.

      Type: INT4
    - **numberOfColumns**
      The number of columns to include in the new matrix.

      Type: INT4
  - Returns
    BOOLEAN TRUE if successful.

Details

This procedure uses drand48_r.

Examples

```
call nzm..uniform('A', 50,50);
call nzm..list_matrices();
call nzm..delete_matrix('A');
```

```
 UNIFORM
---------
 t
(1 row)
 LIST_MATRICES
---------------
 A
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

VEC_TO_DIAG - Create a Diagonal Matrix from a One-column Matrix

This procedure creates the diagonal matrix C using the values stored in the one-column matrix A.

Usage

The VEC_TO_DIAG stored procedure has the following syntax:

- **VEC_TO_DIAG(NVARCHAR(ANY) matrixA, NVARCHAR(ANY) matrixC);**
  - Parameters
    - **matrixA**
      The name of the one-column input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixC**
      The name of the output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Examples

```
call nzm..shape('1,2,3,4',4,1,'A');
call nzm..vec_to_diag('A', 'B');
call nzm..print('A');
call nzm..print('B');
call nzm..delete_matrix('A');
call nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 VEC_TO_DIAG
-------------
 t
(1 row)
```

```
          PRINT
-------------------------
 -- matrix: A --
1
2
3
4
(1 row)
                              PRINT
-------------------------------------------------------------
 -- matrix: B --
1, 0, 0, 0
0, 2, 0, 0
0, 0, 3, 0
0, 0, 0, 4
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

VECDIAG - Diagonal of a Matrix

This procedure extracts the diagonal of a matrix.

Usage

The VECDIAG stored procedure has the following syntax:

- **VECDIAG(matrixAname, matrixCname);**
  - Parameters
    - **matrixAname**
      The name of input matrix A.

      Type: NVARCHAR(ANY)
    - **matrixCname**
      The name of output matrix C.

      Type: NVARCHAR(ANY)
  - Returns
    BOOLEAN TRUE always.

Details

This procedure extracts the diagonal of a matrix using C := diag(A), where A and C are matrices. Matrix A is a square matrix and matrix C is a one column matrix. Matrix C must not exist prior to the operation.

Examples

```
call
nzm..shape('1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16',4,4,'
A');

call nzm..VECDIAG('A','B');

call nzm..print('A');

call nzm..print('B');

call nzm..delete_matrix('A');

call nzm..delete_matrix('B');
```

```
 SHAPE
-------
 t
(1 row)
 VECDIAG
---------
 t
(1 row)
                                   PRINT
```

```
-----------------------------------------------------------------
-----
 -- matrix: A --
1, 2, 3, 4
5, 6, 7, 8
9, 10, 11, 12
13, 14, 15, 16
(1 row)
           PRINT
---------------------------
 -- matrix: B --
1
6
11
16
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
 DELETE_MATRIX
---------------
 t
(1 row)
```

Related Functions

- category matrix operations

Notices and Trademarks

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
26 Forest Street
Marlborough, MA 01752 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

Trademarks

IBM, the IBM logo, ibm.com and Netezza are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™),these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

NEC is a registered trademark of NEC Corporation.

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

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and/or other countries.

D-CC, D-C++, Diab+, FastJ, pSOS+, SingleStep, Tornado, VxWorks, Wind River, and the Wind River logo are trademarks, registered trademarks, or service marks of Wind River Systems, Inc. Tornado patent pending.

APC and the APC logo are trademarks or registered trademarks of American Power Conversion Corporation.

Other company, product or service names may be trademarks or service marks of others.

Regulatory and Compliance

Regulatory Notices

Install the NPS system in a restricted-access location. Ensure that only those trained to operate or service the equipment have physical access to it. Install each AC power outlet near the NPS rack that plugs into it, and keep it freely accessible. Provide approved 30A circuit breakers on all power sources.

Product may be powered by redundant power sources. Disconnect ALL power sources before servicing. High leakage current. Earth connection essential before connecting supply. Courant de fuite élevé. Raccordement à la terre indispensable avant le raccordement au réseau.

Homologation Statement

This product may not be certified in your country for connection by any means whatsoever to interfaces of public telecommunications networks. Further certification may be required by law prior to making any such connection. Contact an IBM representative or reseller for any questions.

FCC - Industry Canada Statement

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio-frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case users will be required to correct the interference at their own expense.

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

CE Statement (Europe)

This product complies with the European Low Voltage Directive 73/23/EEC and EMC Directive 89/336/EEC as amended by European Directive 93/68/EEC.

Warning: This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

VCCI Statement

この装置は、情報処理装置等電波障害自主規制協議会 （VCCI） の基準に基づくクラスＡ 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起越すことがあります。この場合には使用者が適切な対策を講ずるう要求されることがあります。

Index

A

ABS_ELEMENTS,21
ADD,25
ALL_NONZERO,27
ANY_NONZERO,28
APPLY_SIMPLE2RCV_ADV,17

B

BLOCK,30

C

CEIL_ELEMENTS,32
CHOOSE,36
CONCAT,42
COPY_MATRIX,44
COPY_SUBMATRIX,45
COVARIANCE,47
CREATE_IDENTITY_MATRIX,50
CREATE_MATRIX_FROM_TABLE,51
CREATE_ONES_MATRIX,53
CREATE_RANDOM_CAUCHY_MATRIX,54
CREATE_RANDOM_EXPONENT_MATRIX,56
CREATE_RANDOM_GAMMA_MATRIX,58
CREATE_RANDOM_LAPLACE_MATRIX,60
CREATE_RANDOM_MATRIX,62
CREATE_RANDOM_NORMAL_MATRIX,63
CREATE_RANDOM_POISSON_MATRIX,65
CREATE_RANDOM_RAYLEIGH_MATRIX,67
CREATE_RANDOM_UNIFORM_MATRIX,68
CREATE_RANDOM_WEIBULL_MATRIX,70
CREATE_TABLE_FROM_MATRIX,72

D

DEGREES_ELEMENTS,76
DELETE_ALL_MATRICES,79
DELETE_MATRIX,80
DIAG,82
DIVIDE_ELEMENTS,84

E

EIGEN,86
EQ,88
EXP_ELEMENTS,90

F

FLOOR_ELEMENTS,94

G

GE,98
GEMM,100
GET_NUM_COLS,104
GET_NUM_ROWS,105
GET_VALUE,106
GT,107

I

INITIALIZE,109
INSERT,110
INT_ELEMENTS,112
INVERSE,116
IS_INITIALIZED,120

K

KILL_ENGINE,121
KRONECKER,121

L

LE,123
LINEAR_COMBINATION,126
LIST_MATRICES,131
LN_ELEMENTS,132
LOC,136
LOG_ELEMENTS,138
LT,143

M

MATRIX_EXISTS,145
MATRIX_VECTOR_OPERATION,146
MAX,149

MIN,151
MOD_ELEMENTS,153
MTX_LINEAR_REGRESSION,156
MTX_LINEAR_REGRESSION_APPLY,159
MTX_PCA,162
MTX_PCA_APPLY,172
MTX_POW,174
MTX_POW2,176
MULTIPLY_ELEMENTS,178

N

NE,180
NORMAL,182

P

POWER_ELEMENTS,185
PRINT,189

R

RADIANS_ELEMENTS,192
RCV2SIMPLE,196
RCV2SIMPLE_NUM,202
RED_MAX,206
RED_MAX_ABS,207
RED_MIN,208
RED_MIN_ABS,209
RED_SSQ,210
RED_SUM,211
RED_TRACE,212
REDUCE_TO_VECT,213
REDUCTION,215
REMOVE,216
REPEAT,218
ROUND_ELEMENTS,220

S

SCALAR_OPERATION,224
SCALE,229
SET_BLOCK_SIZE,230
SET_GRID_SIZE,231
SET_GRID_SIZE_WITH_REDISTRIBUTE,232
SET_VALUE,233
SHAPE,234
SHAPEMTX,236
SIGN_REVERSE,238
SIMPLE2RCV,242
SIMPLE2RCV_ADV,243
SOLVE,251
SOLVE_LINEAR_LEAST_SQUARES,254
SQRT_ELEMENTS,256
SUBTRACT,259
SVD,261

T

TRANSPOSE,265

U

UNIFORM,267

V

VEC_TO_DIAG,268
VECDIAG,270