SQL Server Matrix Workbench

15 November 2008
by Robyn Page and Phil Factor

```
/*In this workbench, Robyn Page and Phil Factor decide to tackle the subject of
Matrix handling and Matrix Mathematics in SQL. They maintain that 'One just
needs a clear head and think in terms of set-based operations' */
```

```
/* I think that we need to lay to rest the idea that you cannot do matrix operations in SQL. They
are just as easy as they are in a procedural language. one just needs a clear head and think in
terms of set-based operations.
If you're not aware of the usefulness of matrix operations, then you'll be surprised. They
underlie a lot of linear equations and statistics (We do a cool Orthogonal Factor Analysis in
SQL!). They're used a lot in games programming, and bitmap manipulation. We tend to use them for
sticky SQL problems such as reserving seats or timetabling. Once you've started, you'll find a
lot of SQL Problems that can be solved with matrices. (Just in case the formatting goes awry,
we're including the SQL Source file in the speech-bubble above.) Please remember that this isn't
production-quality code: as in all of our workbenches, we're keeping things simple to illustrate
the points as clearly as we can. */

/* we'll create a table to store two-dimensional arrays that contain an exact number with two
digits precision. You'll see that we can create any number of matrixes in this array, which cuts
down on the chore of creating tables. It also means that functions and procedures can be
hardwired to a common matrix table. Of course, you would create the 'element' type according to
your data.*/

IF OBJECT_ID (N'matrix') IS NOT NULL
   DROP TABLE matrix
CREATE TABLE matrix (
       [Name] VARCHAR(10),
       x INT NOT NULL,
       y INT NOT NULL,
       element NUMERIC(9,2) NOT NULL
       PRIMARY KEY ([name],x,y))

/* Yes, this allows the use of sparse arrays, but you would usually need to handle gaps. Note the
compound primary key. This automatically prevents duplication of data in a cell (try it in order
to convice yourself) and provides a reasonable index for most operations. If you use very large
matrixes, you will need to provide an extra covering index. */

/* now let's fill matrix 'A' with some spoof data so we can just do some simple stuff such as
summing the rows and columns */
--we'll choose a ten by fifteen matrix

SET NOCOUNT ON--cut down on the badinage
DELETE FROM matrix WHERE [Name]='A'
DECLARE @ii INT, @jj INT
SELECT @ii=1, @jj=1
WHILE @ii<=10-- loop around as if we were C# programmers!
   BEGIN
   WHILE @jj<=15
       BEGIN
       INSERT INTO matrix SELECT 'A',@ii,@jj,RAND()*200
       SELECT @jj=@jj+1
       END
   SELECT @jj=1, @ii=@ii+1
   END

/* now the simple way to do a summation is by using the built-in grouping functions such as
ROLLUP and CUBE */
SELECT [X/Y]=CASE WHEN GROUPING(y)=0
                  THEN CAST(y AS VARCHAR(6))
                  ELSE 'Sum'
             END,
       '1'=SUM(CASE WHEN x=1 THEN element ELSE 0 END),
       '2'=SUM(CASE WHEN x=2 THEN element ELSE 0 END),
       '3'=SUM(CASE WHEN x=3 THEN element ELSE 0 END),
       '4'=SUM(CASE WHEN x=4 THEN element ELSE 0 END),
       '5'=SUM(CASE WHEN x=5 THEN element ELSE 0 END),
       '6'=SUM(CASE WHEN x=6 THEN element ELSE 0 END),
       '7'=SUM(CASE WHEN x=7 THEN element ELSE 0 END),
       '8'=SUM(CASE WHEN x=8 THEN element ELSE 0 END),
       '9'=SUM(CASE WHEN x=9 THEN element ELSE 0 END),
       '10'=SUM(CASE WHEN x=10 THEN element ELSE 0 END),
```

```
       'total'=SUM(element)
FROM matrix WHERE [name]='a'
GROUP BY y WITH ROLLUP ORDER BY GROUPING(y),y
```

	X/Y	1	2	3	4	5	6	7	8	9	10	total
1	1	25.48	140.39	141.16	47.29	49.97	194.18	119.52	172.06	169.47	191.93	1251.45
2	2	170.38	6.52	46.89	43.26	112.70	65.64	14.05	173.77	122.85	167.57	923.63
3	3	149.12	142.26	112.08	125.57	43.94	175.58	173.73	100.47	83.27	191.42	1297.44
4	4	121.81	92.52	134.44	70.94	192.91	80.18	44.02	146.11	181.02	175.12	1239.07
5	5	63.38	84.43	131.98	185.19	193.01	166.76	73.92	110.91	3.38	194.56	1207.52
6	6	73.11	139.47	94.62	58.81	118.65	46.39	28.13	163.26	16.74	35.91	775.09
7	7	68.38	139.44	62.77	18.21	148.30	112.42	111.34	107.53	168.56	196.65	1133.60
8	8	112.00	54.16	173.21	48.70	79.24	46.63	176.79	174.78	123.79	42.28	1031.58
9	9	59.07	76.13	144.49	66.12	91.47	91.55	54.15	122.37	25.98	96.80	828.13
10	10	180.92	46.59	37.04	28.40	177.47	13.57	179.63	111.16	113.41	168.92	1057.11
11	11	3.67	106.53	191.19	126.57	60.28	193.33	67.77	17.15	70.76	52.03	889.28
12	12	184.64	46.99	187.24	79.59	75.20	153.87	142.17	52.85	30.84	59.80	1013.19
13	13	139.57	179.13	5.32	54.93	139.52	187.09	91.40	11.84	145.76	104.71	1059.27
14	14	50.86	94.02	79.38	27.44	36.58	4.21	78.28	50.48	150.00	139.12	710.37
15	15	83.16	174.10	82.81	56.53	27.55	78.20	70.02	189.97	148.97	23.00	934.31
16	Sum	1485.55	1522.68	1624.62	1037.55	1546.79	1609.60	1424.92	1704.71	1554.80	1839.82	15351.04

```
/* note that, if you are using sparse arrays, you may need to do this.
First, we'll put in a few drips and drabs of data*/
DELETE FROM matrix WHERE [name]='z'
INSERT INTO matrix SELECT 'z',3,5,RAND()*200
INSERT INTO matrix SELECT 'z',7,8,RAND()*200
INSERT INTO matrix SELECT 'z',2,3,RAND()*200
SELECT [X/Y]=CASE WHEN GROUPING(ydimension.number)=0
                  THEN CAST(ydimension.number AS VARCHAR(6))
                  ELSE 'Sum'
             END,
       '1'=SUM(CASE WHEN x=1 THEN element ELSE 0 END),
       '2'=SUM(CASE WHEN x=2 THEN element ELSE 0 END),
       '3'=SUM(CASE WHEN x=3 THEN element ELSE 0 END),
       '4'=SUM(CASE WHEN x=4 THEN element ELSE 0 END),
       '5'=SUM(CASE WHEN x=5 THEN element ELSE 0 END),
       '6'=SUM(CASE WHEN x=6 THEN element ELSE 0 END),
       '7'=SUM(CASE WHEN x=7 THEN element ELSE 0 END),
       '8'=SUM(CASE WHEN x=8 THEN element ELSE 0 END),
       '9'=SUM(CASE WHEN x=9 THEN element ELSE 0 END),
       '10'=SUM(CASE WHEN x=10 THEN element ELSE 0 END),
       'total'=COALESCE(SUM(element),0)
FROM (SELECT number FROM numbers WHERE number <=10) ydimension
LEFT OUTER JOIN  matrix ON ydimension.number =y AND name='z'
GROUP BY ydimension.number WITH ROLLUP
ORDER BY GROUPING(ydimension.number),ydimension.number
```

	X/Y	1	2	3	4	5	6	7	8	9	10	total
1	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	3	0.00	10.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.61
4	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	5	0.00	0.00	79.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	79.43
6	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	8	0.00	0.00	0.00	0.00	0.00	0.00	73.41	0.00	0.00	0.00	73.41
9	9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	Sum	0.00	10.61	79.43	0.00	0.00	0.00	73.41	0.00	0.00	0.00	163.45

```
/* if this is tedious, then use a procedure. Well, of course it's tedious! */
IF OBJECT_ID (N'AggregateMatrix') IS NOT NULL
   DROP PROCEDURE AggregateMatrix
```

```
GO
CREATE PROCEDURE AggregateMatrix
@Name VARCHAR(5)='A', @Aggregation VARCHAR(10)='SUM'
AS
DECLARE @ii INT,@Max INT,@SQL VARCHAR(8000)
SELECT @ii=1,@max=MAX(x) FROM matrix WHERE [name]=@Name
SELECT @SQL='Select [X/Y]=case when grouping(y)=0
then cast(y as varchar(6))
else '''n>+@Aggregation+'''
  end,
'
WHILE @ii<=@max
   SELECT @SQL=@SQL+''''+CAST(@ii AS VARCHAR(5))
                  +'''='+@Aggregation+'(case when x='+CAST(@ii AS VARCHAR(5))
                  +' then element else 0 end),',@ii=@ii+1
SELECT @SQL=@SQL+'''total''='+@Aggregation+'(element)
from matrix where [name]='''+@name+'''
group by y with rollup order by grouping(y),y'
EXECUTE (@SQL)
IF @@error>0 PRINT @SQL
GO

/* and while we're about it, we'll add a procedure to do a simple matrix. This will just be a
cut-down version of the one above */
IF OBJECT_ID (N'SimpleMatrix') IS NOT NULL
   DROP PROCEDURE SimpleMatrix
GO
CREATE PROCEDURE SimpleMatrix
@Name VARCHAR(5)='A'
AS
DECLARE @ii INT,@Max INT,@SQL VARCHAR(8000)
SELECT @ii=1,@max=MAX(x) FROM matrix WHERE [name]=@Name
SELECT @SQL='Select [X/Y]=cast(y as varchar(6))'
WHILE @ii<=@max
   SELECT @SQL=@SQL+',
'''+CAST(@ii AS VARCHAR(5))
       +'''=sum(case when x='+CAST(@ii AS VARCHAR(5))
       +' then element else 0 end)',@ii=@ii+1
SELECT @SQL=@SQL+'
from matrix where [name]='''+@name+''' group by y'

EXECUTE (@SQL)
IF @@error>0 PRINT @SQL
GO


/* OK Let's try them out! */
EXECUTE SimpleMatrix 'a'
```

	X...	1	2	3	4	5	6	7	8	9	10
1	1	25.48	140.39	141.16	47.29	49.97	194.18	119.52	172.06	169.47	191.93
2	2	170.38	6.52	46.89	43.26	112.70	65.64	14.05	173.77	122.85	167.57
3	3	149.12	142.26	112.08	125.57	43.94	175.58	173.73	100.47	83.27	191.42
4	4	121.81	92.52	134.44	70.94	192.91	80.18	44.02	146.11	181.02	175.12
5	5	63.38	84.43	131.98	185.19	193.01	166.76	73.92	110.91	3.38	194.56
6	6	73.11	139.47	94.62	58.81	118.65	46.39	28.13	163.26	16.74	35.91
7	7	68.38	139.44	62.77	18.21	148.30	112.42	111.34	107.53	168.56	196.65
8	8	112.00	54.16	173.21	48.70	79.24	46.63	176.79	174.78	123.79	42.28
9	9	59.07	76.13	144.49	66.12	91.47	91.55	54.15	122.37	25.98	96.80
10	10	180.92	46.59	37.04	28.40	177.47	13.57	179.63	111.16	113.41	168.92
11	11	3.67	106.53	191.19	126.57	60.28	193.33	67.77	17.15	70.76	52.03
12	12	184.64	46.99	187.24	79.59	75.20	153.87	142.17	52.85	30.84	59.80
13	13	139.57	179.13	5.32	54.93	139.52	187.09	91.40	11.84	145.76	104.71
14	14	50.86	94.02	79.38	27.44	36.58	4.21	78.28	50.48	150.00	139.12
15	15	83.16	174.10	82.81	56.53	27.55	78.20	70.02	189.97	148.97	23.00

```
/* In order to do a few common matrix examples, we need to devise a few helper functions to allow
us to get matrix data in there easily, in the format that is easy to check for errors. This is
very handy when you are doing regression testing
```

```
Before we do the routine, let's just make sure you have a number table.... */

IF OBJECT_ID (N'spMaybeBuildNumberTable') IS NOT NULL
   DROP PROCEDURE spMaybeBuildNumberTable
GO
CREATE PROCEDURE spMaybeBuildNumberTable
@size INT=10000 --or whatever size you want to have for your number table
AS
BEGIN
SET NOCOUNT ON
IF NOT EXISTS (SELECT * FROM dbo.sysobjects --check to make sure you have one
  WHERE id = OBJECT_ID(N'[dbo].[Numbers]')
   AND OBJECTPROPERTY(id, N'IsUserTable') = 1)
    BEGIN --create it and stock it.
    CREATE TABLE [dbo].[Numbers](
     [number] [int],
    CONSTRAINT [Index_Numbers] PRIMARY KEY CLUSTERED
    (
     [number] ASC
    ) ON [PRIMARY]
    ) ON [PRIMARY]
   --right lets stock it. Jeff would hate this code.
    DECLARE @ii INT
    SELECT @ii=1
    WHILE (@ii<=@size)
     BEGIN
     INSERT INTO NUMBERS(NUMBER) SELECT @II
     SELECT @II=@II+1
     END
    END
END
/*
Now we can add the function that takes a string representation of a matrix, just like you see in
the textbooks, and returns the SQL Equivalent (Please try it out just to see what it does). It
could be done iteratively, but we have opted for the must faster TSQL-Special technique. If you
are using a different dialect, you'll have to do it the slow iterative way instead, but we're
using SQL Server!
*/
GO
IF OBJECT_ID (N'dbo.MatrixValuesOf') IS NOT NULL
   DROP FUNCTION dbo.MatrixValuesOf
GO
CREATE FUNCTION dbo.MatrixValuesOf(
   @Matrix VARCHAR(MAX),
   @Name VARCHAR(10)
)
RETURNS @MatrixValues TABLE
(
    -- columns returned by the function
    name VARCHAR(10) NOT NULL,
    x INT NOT NULL,
    y INT NOT NULL,
    [Value] NUMERIC(8,2)
)
AS
BEGIN
DECLARE @y INT, @x INT, @Value VARCHAR(200)
DECLARE @MyTable TABLE (TheOrder INT PRIMARY KEY,
                        TheChar CHAR(1) NOT NULL,x INT,y INT,
                        Thevalue NUMERIC(8,2),
                        strvalue VARCHAR(200))
SELECT @x=1, @y=1,@Value=''
INSERT INTO @MyTable (TheOrder,TheChar)
   SELECT number, SUBSTRING(@Matrix,number,1)
   FROM numbers WHERE number<=LEN(@Matrix)
UPDATE @MyTable
   SET @x =x= CASE WHEN TheChar=',' THEN @x+1 WHEN TheChar=CHAR(13) THEN 1 ELSE @x END,
       @y =y= CASE WHEN TheChar=CHAR(13) THEN @y+1 ELSE @y END,
        @value=strvalue=CASE WHEN TheChar IN (',',CHAR(13)) THEN ''
               WHEN '1234567890-+.' LIKE '%'+theChar+'%' THEN @value+TheChar ELSE @value END
INSERT INTO @MatrixValues
SELECT @Name, x, y, CAST(RTRIM(LTRIM(CASE WHEN ISNUMERIC(strValue)=1 THEN strValue ELSE '0' END))
AS NUMERIC(8,2))
   FROM @MyTable WHERE TheOrder IN (SELECT MAX(TheOrder) FROM @MyTable GROUP BY x,y)
RETURN
END
```

```
GO
--so let's test it out now....
DECLARE @matrixA VARCHAR(2000)
SELECT @MatrixA=
'2,  34,  67.78
 4,   5,   7
 5, 457,   8.65'
SELECT * FROM MatrixValuesOf(@MatrixA,'a')

/* Wee!! that worked, didn't it!
name       x           y           Value
---------- ----------- ----------- --------
a          1           1           2.00
a          2           1           34.00
a          3           1           67.78
a          1           2           4.00
a          2           2           5.00
a          3           2           7.00
a          1           3           5.00
a          2           3           457.00
a          3           3           8.65
*/

--Let's double-check...
DELETE FROM matrix WHERE [Name]='Z'
INSERT INTO matrix
SELECT * FROM MatrixValuesOf(
'5.31, 1.54, 0.09, 3.60, 8.64, 1.01, 8.05, 3.59, 2.61, 8.59
 4.61, 2.29, 9.16, 4.19, 8.17, 0.29, 6.92, 9.96, 9.00, 3.68
 8.56, 9.67, 6.40, 4.25, 2.39, 7.30, 0.42, 2.62, 3.73, 2.54
 8.86, 6.47, 1.06, 6.98, 3.54, 1.05, 2.93, 1.85, 4.35, 4.11
 4.14, 1.64, 2.35, 2.11, 6.08, 6.52, 3.07, 9.62, 0.67, 8.32'
,'z')
EXECUTE SimpleMatrix 'z'
```

	X...	1	2	3	4	5	6	7	8	9	10
1	1	5.31	1.54	0.09	3.60	8.64	1.01	8.05	3.59	2.61	8.59
2	2	4.61	2.29	9.16	4.19	8.17	0.29	6.92	9.96	9.00	3.68
3	3	8.56	9.67	6.40	4.25	2.39	7.30	0.42	2.62	3.73	2.54
4	4	8.86	6.47	1.06	6.98	3.54	1.05	2.93	1.85	4.35	4.11
5	5	4.14	1.64	2.35	2.11	6.08	6.52	3.07	9.62	0.67	8.32

```
/* Now we can get started!
```

matrix multiplication by a scalar

```
*/
DELETE FROM matrix WHERE [Name]='s'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
' 4,   5, -56
 14,  11,  43
-43,  -4,  62' ,
's')
INSERT INTO matrix
   SELECT 't',x,y,element*2
   FROM matrix s
   WHERE s.[name]='s'
--and let's see the result....
EXECUTE SimpleMatrix 't'
/* so scalar arithmetic is absurdly simple and need concern us no longer */
```

	X/Y	1	2	3
1	1	8.00	10.00	-112.00
2	2	28.00	0.00	86.00
3	3	-86.00	-8.00	124.00

```
/*
```

Matrix Addition

```
 Here is how we do Matrix Addition */

--pop in Matrix A
DELETE FROM matrix WHERE [Name]='a'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
'10, 13, -2
 12,  1,  4
 -4, -4,  6' ,
'a')

--add in Matrix B
DELETE FROM matrix WHERE [Name]='b'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
' 3, 24,  2
 -4, 14, 46
  1,  3, -5' ,
'b')
--now create a matrix 'c' which is the matrix addition of A and B
DELETE FROM matrix WHERE [Name]='c'
INSERT INTO matrix
   SELECT 'c',a.x,a.y,a.element+b.element
   FROM matrix a
   INNER JOIN matrix b
   ON a.x=b.x
   AND a.y=b.y
   WHERE a.[name]='a' AND b.[name]='b'
--and let's see the result....
EXECUTE SimpleMatrix 'c'
/*
X/Y    1             2             3
------ ------------- ------------- -------------
1      13.00         37.00          0.00
2       8.00         15.00         50.00
3      -3.00         -1.00          1.00

*/
/*
```

Matrix subtraction

```
 and matrix subtraction is just as easy */
DELETE FROM matrix WHERE [Name]='d'
INSERT INTO matrix
   SELECT 'd',a.x,a.y,a.element-b.element
   FROM matrix a
   INNER JOIN matrix b
   ON a.x=b.x
   AND a.y=b.y
   WHERE a.[name]='a' AND b.[name]='b'
/* and now let's see it! */
EXECUTE SimpleMatrix 'd'
/*
X/Y    1             2             3
------ ------------- ------------- -------------
1       7.00         -11.00         -4.00
2      16.00         -13.00        -42.00
3      -5.00          -7.00         11.00

*/

/*
```

Matrix Multiplication

```
So you think you can do the same thing with matrix multiplication eh? Just use a '*' instead of a
'-'? Think again. this is where it gets more complicated
We'll multiply two nice simple matrices...*/

DELETE FROM matrix WHERE name IN ('e','f')
INSERT INTO matrix
```

```
   SELECT * FROM MatrixValuesOf(
' 1, 2, 3
  4, 5, 6' ,
'e')

INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
' 7,  8
  9, 10
  11,12' ,
'f')

/* create the result of the matrix multiplication */
INSERT INTO matrix
SELECT 'g',xf,ye,SUM(ee*ef)
   FROM
   (SELECT [ee]=e.element, [ef]=f.element,[xe]=e.x,[ye]=e.y,[xf]=f.x,[yf]=f.y
   FROM matrix e
   INNER JOIN
      Matrix f
   ON e.x=f.y
   WHERE e.[name]='e' AND f.[name]='f')pairs
GROUP BY ye,xf

EXECUTE SimpleMatrix 'g'
```

	X/Y	1	2
1	1	58.00	64.00
2	2	139.00	154.00

Matrix Horizontal concatenation

```
--horizontal concatenation is cool
--pop in Matrix A
DELETE FROM matrix WHERE [Name]='a'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
'10, 13, -2
12,  1,  4
-4, -4,  6' ,
'a')
--add in Matrix B
DELETE FROM matrix WHERE [Name]='b'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
' 3, 24,  2
-4, 14, 46
  1,  3, -5' ,
'b')
--now create a matrix 'c' which is A and B concatenated ( normally, you'd put the MAX value into
a variable, of course but we wanted to keep it simple)
DELETE FROM matrix WHERE [Name]='c'
INSERT INTO matrix
   SELECT 'c', [x]=CASE name WHEN 'a' THEN x ELSE x+(SELECT MAX(x) FROM matrix WHERE [name]='a')
END,y, element
FROM matrix t WHERE t.[name] IN ('a','b')

EXECUTE SimpleMatrix 'c'
```

	X/Y	1	2	3	4	5	6
1	1	10.00	13.00	-2.00	3.00	24.00	2.00
2	2	12.00	1.00	4.00	-4.00	14.00	46.00
3	3	-4.00	-4.00	6.00	1.00	3.00	-5.00

```
/*
```

Matrix Vertical Concatenation

```
 or we can do it vertically (sentiment about the variable applies here too */
```

```
DELETE FROM matrix WHERE [Name]='c'
INSERT INTO matrix
   SELECT 'c', [x]=x,[y]=CASE name WHEN 'a' THEN y ELSE y+(SELECT MAX(y) FROM matrix WHERE [name]
='a') END, element
FROM matrix t WHERE t.[name] IN ('a','b')

EXECUTE SimpleMatrix 'c'
```

	X...	1	2	3
1	1	10.00	13.00	-2.00
2	2	12.00	1.00	4.00
3	3	-4.00	-4.00	6.00
4	4	3.00	24.00	2.00
5	5	-4.00	14.00	46.00
6	6	1.00	3.00	-5.00

Matrix Transposition

```
 Fine, so how do we do Matrix transposition? */

DELETE FROM matrix WHERE [Name]='t'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
'1,  2,  3
4,  5,  6' ,
't')
DELETE FROM matrix WHERE [Name]='u'
INSERT INTO matrix
SELECT 'u', [x]=y, [y]=x, element
FROM matrix t WHERE t.[name]='t'

EXECUTE SimpleMatrix 'u'--and display the transposed matrix
```

	X...	1	2
1	1	1.00	4.00
2	2	2.00	5.00
3	3	3.00	6.00

```
/*
```

Rendering a matrix as a string

```
 So now, just to finish off with a flourish, we'll reverse-out our 'MatrixValuesOf' Function, so
we can represent the results in a string. This means that we could make an easier test harness
and could squirrel matrixes away in string form. */

IF OBJECT_ID (N'StringMatrix') IS NOT NULL
   DROP PROCEDURE StringMatrix
GO
CREATE PROCEDURE StringMatrix
@Name VARCHAR(5)='A',
@String VARCHAR(MAX) OUTPUT
AS
SET NOCOUNT ON
DECLARE @ii INT,@Max INT,@SQL NVARCHAR(4000)
SELECT @ii=1,@max=MAX(x) FROM matrix WHERE [name]=@Name
WHILE @ii<=@max
   SELECT @SQL=COALESCE(@SQL+'
              +','Select @string=coalesce(@String,'''')+')+'right(''          ''+cast(cast(sum
(case when x='+CAST(@ii AS VARCHAR(5))
       +' then element else 0 end) as numeric(9,2)) as varchar(12)),10)'+CASE WHEN @ii<@max THEN
'+'','''' ELSE '+''
''' END,@ii=@ii+1
SELECT @SQL=@SQL+'
from matrix where [name]='''+@name+''' group by y'
EXECUTE sp_ExecuteSQL @SQL,N' @String varchar(max) output', @String=@String OUTPUT
```

```
IF @@error>0 PRINT @SQL
GO

DECLARE @String VARCHAR(MAX)
EXECUTE StringMatrix 'a',@String OUTPUT
SELECT @String

/*   10.00,     13.00,     -2.00
     12.00,      1.00,      4.00
     -4.00,     -4.00,      6.00

So we are ready for a more exacting test... */
DELETE FROM matrix WHERE [Name]='x'
INSERT INTO matrix
   SELECT * FROM MatrixValuesOf(
'   192.96,    -55.82,     92.33,     55.86,     58.47,     11.05,     53.26
    192.08,    177.95,    167.00,    196.71,     79.70,     91.51,     58.13
     12.00,     54.32,    104.72,     63.03,    183.26,    169.45,    117.58
    102.87,    189.95,     19.50,    170.58,      0.01,     80.48,    146.79
    171.93,     62.35,     88.21,     54.73,     91.39,    151.02,    175.50
     44.19,    150.64,    196.40,     17.24,    133.31,    102.52,    146.05
    199.20,    190.81,    -38.56,     53.04,     47.94,    178.62,     48.57
    104.10,     41.95,    175.17,   -100.81,     28.55,     82.87,     38.74
    126.57,     26.02,     20.88,     59.68,     82.01,     92.68,    119.01
    184.44,     27.17,     85.33,    139.79,    123.22,    112.38,     38.28
    195.55,      3.82,    170.43,    170.51,    149.36,    118.25,     78.22
    135.32,     36.93,     28.97,   -111.45,    168.44,     68.15,    189.48
     84.04,    107.10,     19.77,      3.20,     69.30,    175.64,     29.29
     16.90,     42.28,     73.41,    154.76,     81.55,     57.64,     96.12
     66.61,     37.63,     62.82,     87.71,    102.84,    110.43,    185.25'
,'x')
GO
DECLARE @String VARCHAR(MAX)
EXECUTE StringMatrix 'x',@String OUTPUT
SELECT @String
/* which produces this! (phew)

    192.96,    -55.82,     92.33,     55.86,     58.47,     11.05,     53.26
    192.08,    177.95,    167.00,    196.71,     79.70,     91.51,     58.13
     12.00,     54.32,    104.72,     63.03,    183.26,    169.45,    117.58
    102.87,    189.95,     19.50,    170.58,      0.01,     80.48,    146.79
    171.93,     62.35,     88.21,     54.73,     91.39,    151.02,    175.50
     44.19,    150.64,    196.40,     17.24,    133.31,    102.52,    146.05
    199.20,    190.81,    -38.56,     53.04,     47.94,    178.62,     48.57
    104.10,     41.95,    175.17,   -100.81,     28.55,     82.87,     38.74
    126.57,     26.02,     20.88,     59.68,     82.01,     92.68,    119.01
    184.44,     27.17,     85.33,    139.79,    123.22,    112.38,     38.28
    195.55,      3.82,    170.43,    170.51,    149.36,    118.25,     78.22
    135.32,     36.93,     28.97,   -111.45,    168.44,     68.15,    189.48
     84.04,    107.10,     19.77,      3.20,     69.30,    175.64,     29.29
     16.90,     42.28,     73.41,    154.76,     81.55,     57.64,     96.12
     66.61,     37.63,     62.82,     87.71,    102.84,    110.43,    185.25


So, what goes in comes out (it took a little bit of heaving and grunting). So this is as far as
we'll take it in this workbench and we'll leave you to figure out a few little puzzles such as
inversion and determinants. Now, the next puzzle is how one can use these techniques to solve
other programming puzzles. It would be fascinating TO learn how this sort OF technique can be
used.
*/
```