

The Configured Data Masks with Dynamic Data Masking

In this article we will look at the other types of data masking options available to us. We will not examine the behaviour of every data type for each mask. However we will look at the behaviour of many how we might expect the various masks to work.

This article will examine the three masks coming in SQL Server 2016 outside of the default data mask. These are the

- Email
- Random
- Custom String (or pattern)

The Email Mask

Let's start with the email mask. This is one of the more common patterns that applications often mask. The email mask results in a mask that works as follows:

- The first letter of the data (the email address)
- A series of x's before the @ symbol.
- A series of x's for the domain and subdomain(s)
- A .com at the end.

Let's see how this works in practice. Let's create a table and add some data. Note that this mask uses the word email, with open and close parenthesis. No parameters are used in the SQL Server 2016 version.

```
CREATE TABLE dbo.Contacts
(ContactID INT
, ContactName VARCHAR(200)
, ContactEmail VARCHAR(300) MASKED WITH (FUNCTION='email()')
)
GO
INSERT dbo.Contacts
VALUES
(1, 'Malapati Bharath', 'bharath.malapati@fiserv.com')
, (2, 'Akhtar Rehan', 'Rehan.Akhtar@Fiserv.com')
, (3, 'John', 'John@volleyball.rec')
, (4, 'Dan', 'dan@knowledge.co.us.edu')
, (5, 'Matt', 'Matt@live.co.nz')
```

If we query this data as a non-privileged user, we find these results:

```
CREATE USER cmtest WITHOUT LOGIN
GRANT SELECT ON [dbo].[Contacts] TO cmtest
GO

-- Query as the test user
EXECUTE AS USER = 'cmtest'
GO
```

```

SELECT
*
FROM
    dbo.Contacts
GO
REVERT
GO

```

	ContactID	ContactName	ContactEmail
1	1	Malapati Bharath	bXXX@XXXX.com
2	2	Akhtar Rehan	RXXX@XXXX.com
3	3	John	JXXX@XXXX.com
4	4	Dan	dXXX@XXXX.com
5	5	Matt	MXXX@XXXX.com

As you can see, we get a consistent mask. The size, length, format, etc. of the data is irrelevant. We always get a one character beginning to the email, a set of 3 Xs, the @ symbol, 4 Xs, and then ".com." Even if I have other domains, or multiple subdomains, the mask is consistent and doesn't reveal other domains.

Let's now add some other data and see how this affects the mask. Let's add some malformed domains and problem emails.

```

INSERT dbo.Contacts
VALUES
    (6, 'James', 'james.com')
, (7, 'Galvin', null)
, (8, 'Nick', 'dog.org')
, (9, 'Deuce', 'DeucesAreWild')
, (10, 'Oscar', 'grouch@')
GO

```

If we query this data as a non-privileged user, we find these results:

	ContactID	ContactName	ContactEmail
1	1	Malapati Bharath	bXXX@XXXX.com
2	2	Akhtar Rehan	RXXX@XXXX.com
3	3	John	JXXX@XXXX.com
4	4	Dan	dXXX@XXXX.com
5	5	Matt	MXXX@XXXX.com
6	6	James	jXXX@XXXX.com
7	7	Galvin	NULL
8	8	Nick	dXXX@XXXX.com
9	9	Deuce	DXXX@XXXX.com
10	10	Oscar	gXXX@XXXX.com

It doesn't matter how the data is formed. Only the NULL reveals there is NULL data in that field. We can see that even the malformed emails, for "dog.org" and "grouch@" are properly masked.

The email mask is limited to:

- char
- nchar
- varchar (including max)
- nvarchar (including max)
- text
- ntext

The email mask is built for strings. If you attempt to mask any of the other types with the email mask, you receive an error on table create/alter.

The Random Mask

The random mask is designed for use with numeric types. The idea is that a random number is used to replace the existing data with something else. The RANDOM mask takes two parameters, which are the start and end of the range. The format is shown below.

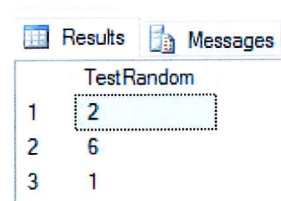
Let's run some tests here. We will create a new table here:

```
CREATE TABLE dbo.RandomTest
( TestRandom int MASKED WITH (FUNCTION='random(1, 10)')
)
GO
INSERT dbo.RandomTest
VALUES
(100), (200), (300)
GO
```

If we query this data as a non-privileged user, we find these results:

```
GRANT SELECT ON [dbo].[RandomTest] TO cmtest
GO
```

```
-- Query as the test user
EXECUTE AS USER = 'cmtest'
GO
SELECT
*
FROM
dbo.RandomTest
GO
REVERT
GO
```



The screenshot shows a SQL Server query results window with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with the following data:

	TestRandom
1	2
2	6
3	1

Let's run this multiple times, here are 3 consecutive executions.

TestRandom	
1	3
2	6
3	9

TestRandom	
1	5
2	5
3	3

TestRandom	
1	5
2	4
3	2

As you can see, the same query returns new data each time.

Let's run some more tests here. If we choose a type like numeric or float, we get values that fit in those types. We will create a new table here:

```
CREATE TABLE dbo.RandomNumericTest
(TestNumericRandom NUMERIC(10,2) MASKED WITH (FUNCTION='random(200, 500)')
)
GO
INSERT dbo.RandomNumericTest
VALUES
(23.32), (101.08), (999.99)
GO
```

If we query this data as a non-privileged user, we find these results:

TestNumericRandom	
1	447.50
2	411.36
3	389.29

If we run this multiple times, we will get a variety of values.

The random mask is built for numeric data types.

The Custom String Mask

The custom string mask uses the function "partial". Here we specify the function as:

Partial (prefix, [padding], suffix)

We expose the first few letters of the data, based on the value of prefix. Prefix needs to be a number. The same thing for suffix, where the last characters are exposed. If the original value is too short, the prefix/suffix is not exposed. The middle value is the actual mask to use. This is a string. Note that this value is enclosed in double quotes.

In this case, we only expose 2 characters of the prefix, 4 of the suffix, and a "xxx.xxx" in between. Here are the DDL and insert statements.

```
CREATE TABLE dbo.ContactsPartialTest
( ContactID INT
, ContactName VARCHAR(200)
, ContactEmail VARCHAR(300) MASKED WITH (FUNCTION='partial(2, "xxx.xxx", 4)')
)
GO
INSERT dbo.ContactsPartialTest
VALUES
```

```

    (1, 'Malapati Bharath', 'bharath.malapati@fiserv.com')
, (2, 'Akhtar Rehan', 'Rehan.Akhtar@Fiserv.com')
, (3, 'John', 'John@volleyball.rec')
, (4, 'Dan', 'dan@knowledge.co.us.edu')
, (5, 'Matt', 'Matt@live.co.nz')
GO

```

If we query this data as a non-privileged user, we find these results:

```

GRANT SELECT ON [dbo].[ContactsPartialTest] TO cmtest
GO

-- Query as the test user
EXECUTE AS USER = 'cmtest'
GO
SELECT
    *
FROM
    dbo.ContactsPartialTest
GO
REVERT
GO

```



	ContactID	ContactName	ContactEmail
1	1	Malapati Bharath	bhxxx.xxx.com
2	2	Akhtar Rehan	Rxxxx.xxx.com
3	3	John	Jxxxx.xxx.rec
4	4	Dan	daxxx.xxx.edu
5	5	Matt	Maxxx.xxxx.nz

As we can see, this is a more realistic masking of the actual data, which could be useful in situations where the user needs to see a portion, but a very limited portion of data.

The custom string mask is built for string data types.

Conclusion

In this article, we have examined the Email, Random, and Custom String masks. Two of these are built for string types, and one for numeric types. Neither of these masks supports dates or the other specialized types. These are limitations, but since the default mask supports most types, we can mask data if needed.

We can expect the types of functions and masking to grow over time, potentially even allowing us to write SQLCLR masks, maybe User Defined Data Masks (UDDM) at some point. In any case, this is a good addition to SQL Server and a nice set of functions to use.

Once again, remember this isn't a security function, and there could be data leakage. Treat this as an application programming convenience feature that simplifies the masking process in development.