**Obfuscate column level data by using “Dynamic Data Masking” in SQL Server 2016**

This time **SQL Server 2016** has made good additions in area of Security by introducing features like:

**1. Always Encrypted**

**2. Row Level Security**

**3. Dynamic Data Masking**, this post

**4.** and other security features, like **Transparent Data Encryption** (TDE), etc.

**Dynamic Data Masking** provides you support for **real-time obfuscation** of data so that the data requesters do not get access to unauthorized data. This helps protect sensitive data even when it is not encrypted, and shows obfuscated data at the presentation layer without changing anything at the database level.

**Dynamic Data Masking** limits sensitive data exposure by masking it to **non-privileged users**. This feature helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal impact on the application layer. It’s a **Policy-based Security** feature that hides the sensitive data in the result set of a Query over designated database columns, while the data in the database is not changed.

**–> “Dynamic Data Masking” provides you three functions/options to Mask your data:**

**1. default():** just replaces the column value with ‘XXXX’ by default.

**2. email():** shows an email ID to this format ‘aXXX@XXXX.com’.

**3. partial(prefix,padding,suffix):** gives you option to format and mask only some part of a string value.

We will see the usage of all these 3 masking functions below.

**–> To setup Dynamic Data Masking on a particular Table you need to:**

1. **CREATE TABLE** with **MASKED WITH FUNCTION** option at column level.

– Or **ALTER TABLE** columns by using this option if the table is already present.

2. Create Users and Grant Read/SELECT access for the above CREATED/ALTERED table.

**–> 1. Create a sample table [dbo].[Customer] with masked columns:**

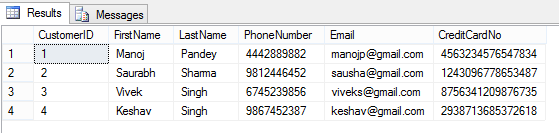
|  |  |
| --- | --- |
|  | CREATE TABLE dbo.Customer (      CustomerID   INT IDENTITY PRIMARY KEY,      FirstName    VARCHAR(250),      LastName     VARCHAR(250) MASKED WITH (FUNCTION = 'default()') NULL,      PhoneNumber  VARCHAR(12)  MASKED WITH (FUNCTION = 'partial(1,"XXXXXXXXX",0)') NULL,      Email        VARCHAR(100) MASKED WITH (FUNCTION = 'email()') NULL,      CreditCardNo VARCHAR(16)  MASKED WITH (FUNCTION = 'partial(4,"XXXXXXXXXX",2)') NULL,  ); |

**–> Now insert some test records (fictitious figures):**

|  |  |
| --- | --- |
|  | INSERT INTO dbo.Customer (FirstName, LastName, PhoneNumber, Email, CreditCardNo)  VALUES  ('Manoj',   'Pandey', '4442889882', 'manojp@gmail.com', '4563234576547834'),  ('Saurabh', 'Sharma', '9812446452', 'sausha@gmail.com', '1243096778653487'),  ('Vivek',   'Singh',  '6745239856', 'viveks@gmail.com', '8756341209876735'),  ('Keshav',  'Singh',  '9867452387', 'keshav@gmail.com', '2938713685372618'); |

**–> Let’s check the rows on [dbo].[Customer] table in context of my User:**

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM dbo.Customer;  GO |

[](https://sqlwithmanoj.files.wordpress.com/2015/07/sqlserver2016-ddm-01.png)

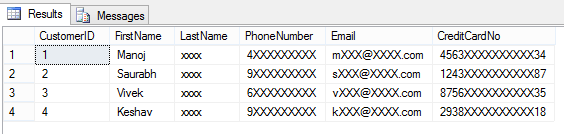
Here, I can see all the row/column values as I have full access to read the masked/sensitive data.

**–> 2.a. Now let’s create a Test Account and just Grant Read access to [dbo].[Customer] table:**

|  |  |
| --- | --- |
|  | CREATE USER AnyUser WITHOUT LOGIN;  GO    GRANT SELECT ON dbo.Customer TO AnyUser;  GO |

**–> Let’s execute the SELECT statement on [dbo].[Customer] table in the Context of this new user account:**

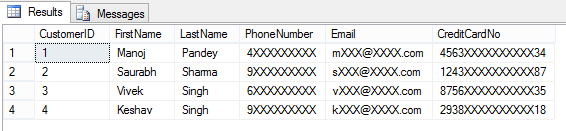
|  |  |
| --- | --- |
|  | EXECUTE AS USER = 'AnyUser';  SELECT \* FROM dbo.Customer;  REVERT;  GO |

[](https://sqlwithmanoj.files.wordpress.com/2015/07/sqlserver2016-ddm-02.png)

And you can see that the user is not able to see the masked data as he is not authorized to see it.

**–> Removing Masking from a column by simple ALTER TABLE/COLUMN statement:**

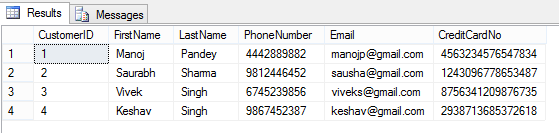
|  |  |
| --- | --- |
|  | ALTER TABLE dbo.Customer  ALTER COLUMN LastName DROP MASKED;  GO    -- Let's check the table data again:  EXECUTE AS USER = 'AnyUser';  SELECT \* FROM dbo.Customer;  REVERT;  GO |

[](https://sqlwithmanoj.files.wordpress.com/2015/07/sqlserver2016-ddm-03.png)

Here, now you are able to see contents of LastName columns, as the masking has been removed from this column by using simple ALTER TABLE/COLUMN statement.

**–> 2.b. Granting** the **UNMASK permission** to “AnyUser”:

|  |  |
| --- | --- |
|  | GRANT UNMASK TO AnyUser;  GO    -- Let's check the table data again:  EXECUTE AS USER = 'AnyUser';  SELECT \* FROM dbo.Customer;  REVERT;  GO |

[](https://sqlwithmanoj.files.wordpress.com/2015/07/sqlserver2016-ddm-01.png)

He is able to see all data unmasked when the **UNMASK permission** is granted to this user.

**–> 2.c. Revoking** back the **UNMASK permission** form the same user:

|  |  |
| --- | --- |
|  | REVOKE UNMASK TO AnyUser;  GO  EXECUTE AS USER = 'AnyUser';  SELECT \* FROM dbo.Customer;  REVERT;  GO |

After Revoking UNMASK permission he is again not able to see complete data.

This way you can control access to your precious or PII data by masking the columns/fields that you don’t want to show to the external world or some set of users.

**–> Final Cleanup:**

|  |  |
| --- | --- |
|  | DROP TABLE dbo.Customer;  GO  DROP USER [AnyUser]  GO |