<https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16>

# Dynamic data masking

* Article
* 04/15/2024
* 30 contributors

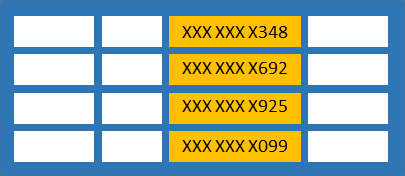
Feedback

## In this article

1. [Overview of dynamic data masking](https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16#overview-of-dynamic-data-masking)
2. [Define a dynamic data mask](https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16#define-a-dynamic-data-mask)
3. [Permissions](https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16#permissions)
4. [Best practices and common use cases](https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16#best-practices-and-common-use-cases)

Show 6 more

**Applies to:**  SQL Server 2016 (13.x) and later  [Azure SQL Database](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)  [Azure SQL Managed Instance](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)  [Azure Synapse Analytics](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)



Dynamic data masking (DDM) limits sensitive data exposure by masking it to nonprivileged users. It can be used to greatly simplify the design and coding of security in your application.

This content applies to dynamic data masking concepts generally, and specific to SQL Server. Content specific to other platforms is available:

* For dynamic data masking in Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics, see [Get started with SQL Database Dynamic Data Masking](https://learn.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview).
* For dynamic data masking in Microsoft Fabric, see [Dynamic data masking in Fabric data warehousing](https://learn.microsoft.com/en-us/fabric/data-warehouse/dynamic-data-masking).

## Overview of dynamic data masking

Dynamic data masking helps prevent unauthorized access to sensitive data by enabling customers to specify how much sensitive data to reveal with minimal effect on the application layer. DDM can be configured on designated database fields to hide sensitive data in the result sets of queries. With DDM, the data in the database isn't changed. DDM is easy to use with existing applications, since masking rules are applied in the query results. Many applications can mask sensitive data without modifying existing queries.

* A central data masking policy acts directly on sensitive fields in the database.
* Designate privileged users or roles that do have access to the sensitive data.
* DDM features full masking and partial masking functions, and a random mask for numeric data.
* Simple Transact-SQL commands define and manage masks.

The purpose of dynamic data masking is to limit exposure of sensitive data, preventing users who shouldn't have access to the data from viewing it. Dynamic data masking doesn't aim to prevent database users from connecting directly to the database and running exhaustive queries that expose pieces of the sensitive data. Dynamic data masking is complementary to other SQL Server security features (auditing, encryption, row level security, etc.) and it's highly recommended to use it with them in order to better protect the sensitive data in the database.

Dynamic data masking is available in SQL Server 2016 (13.x) and Azure SQL Database, and is configured by using Transact-SQL commands. For more information about configuring dynamic data masking by using the Azure portal, see [Get started with SQL Database Dynamic Data Masking (Azure portal)](https://learn.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview).

**Note**

[**Microsoft Entra ID**](https://learn.microsoft.com/en-us/entra/fundamentals/new-name) was previously known as Azure Active Directory (Azure AD).

## Define a dynamic data mask

A masking rule can be defined on a column in a table, in order to obfuscate the data in that column. Five types of masks are available.

Expand table

| **Function** | **Description** | **Examples** |
| --- | --- | --- |
| Default | Full masking according to the data types of the designated fields.  For string data types, use XXXX (or fewer) if the size of the field is fewer than 4 characters (**char**, **nchar**, **varchar**, **nvarchar**, **text**, **ntext**).  For numeric data types use a zero value (**bigint**, **bit**, **decimal**, **int**, **money**, **numeric**, **smallint**, **smallmoney**, **tinyint**, **float**, **real**).  For date and time data types, use 1900-01-01 00:00:00.0000000 (**date**, **datetime2**, **datetime**, **datetimeoffset**, **smalldatetime**, **time**).  For binary data types use a single byte of ASCII value 0 (**binary**, **varbinary**, **image**). | Example column definition syntax: Phone# varchar(12) MASKED WITH (FUNCTION = 'default()') NULL  Example of alter syntax: ALTER COLUMN Gender ADD MASKED WITH (FUNCTION = 'default()') |
| Email | Masking method that exposes the first letter of an email address and the constant suffix ".com", in the form of an email address. aXXX@XXXX.com. | Example definition syntax: Email varchar(100) MASKED WITH (FUNCTION = 'email()') NULL  Example of alter syntax: ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()') |
| Random | A random masking function for use on any numeric type to mask the original value with a random value within a specified range. | Example definition syntax: Account\_Number bigint MASKED WITH (FUNCTION = 'random([start range], [end range])')  Example of alter syntax: ALTER COLUMN [Month] ADD MASKED WITH (FUNCTION = 'random(1, 12)') |
| Custom String | Masking method that exposes the first and last letters and adds a custom padding string in the middle. prefix,[padding],suffix  If the original value is too short to complete the entire mask, part of the prefix or suffix isn't exposed. | Example definition syntax: FirstName varchar(100) MASKED WITH (FUNCTION = 'partial(prefix,[padding],suffix)') NULL  Example of alter syntax: ALTER COLUMN [Phone Number] ADD MASKED WITH (FUNCTION = 'partial(1,"XXXXXXX",0)')  This turns a phone number like 555.123.1234 into 5XXXXXXX.  Additional example:  ALTER COLUMN [Phone Number] ADD MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",0)')  This turns a phone number like 555.123.1234 into 555.1XXXXXXX. |
| Datetime | **Applies to:** SQL Server 2022 (16.x)  Masking method for column defined with data type **datetime**, **datetime2**, **date**, **time**, **datetimeoffset**, **smalldatetime**. It helps masking the year => datetime("Y"), month=> datetime("M") , day=>datetime("D"), hour=>datetime("h"), minute=>datetime("m"), or seconds=>datetime("s") portion of the day. | Example of how to mask the year of the **datetime** value:  ALTER COLUMN BirthDay ADD MASKED WITH (FUNCTION = 'datetime("Y")')  Example of how to mask the month of the **datetime** value:  ALTER COLUMN BirthDay ADD MASKED WITH (FUNCTION = 'datetime("M")')  Example of how to mask the minute of the **datetime** value:  ALTER COLUMN BirthDay ADD MASKED WITH (FUNCTION = 'datetime("m")') |

## Permissions

Users with **SELECT** permission on a table can view the table data. Columns that are defined as masked display the masked data. Grant the **UNMASK** permission to a user to allow them to retrieve unmasked data from the columns for which masking is defined.

Administrative users and roles can always view unmasked data via the **CONTROL** permission, which includes both the **ALTER ANY MASK** and **UNMASK** permission. Administrative users or roles such as sysadmin or db\_owner have CONTROL permissions on the database by design, and can view unmasked data.

You don't need any special permission to create a table with a dynamic data mask, only the standard **CREATE TABLE** and **ALTER** on schema permissions.

Adding, replacing, or removing the mask of a column, requires the **ALTER ANY MASK** permission and **ALTER** permission on the table. It's appropriate to grant **ALTER ANY MASK** to a security officer.

**Note**

The UNMASK permission does not influence metadata visibility: granting UNMASK alone doesn't disclose any metadata. UNMASK will always need to be accompanied by a SELECT permission to have any effect. Example: granting UNMASK on database scope and granting SELECT on an individual Table will have the result that the user can only see the metadata of the individual table from which he can select, not any others. Also see [**Metadata Visibility Configuration**](https://learn.microsoft.com/en-us/sql/relational-databases/security/metadata-visibility-configuration?view=sql-server-ver16).

## Best practices and common use cases

* Creating a mask on a column doesn't prevent updates to that column. So although users receive masked data when querying the masked column, the same users can update the data if they have write permissions. A proper access control policy should still be used to limit update permissions.
* Using SELECT INTO or INSERT INTO to copy data from a masked column into another table results in masked data in the target table (assuming it's exported by a user without **UNMASK** privileges).
* Dynamic Data Masking is applied when running SQL Server Import and Export. A database containing masked columns results in an exported data file with masked data (assuming it's exported by a user without **UNMASK** privileges), and the imported database will contain statically masked data.

## Query for masked columns

Use the sys.masked\_columns view to query for table-columns that have a masking function applied to them. This view inherits from the sys.columns view. It returns all columns in the sys.columns view, plus the is\_masked and masking\_function columns, indicating if the column is masked, and if so, what masking function is defined. This view only shows the columns on which there's a masking function applied.

SQLCopy

SELECT c.name, tbl.name as table\_name, c.is\_masked, c.masking\_function

FROM sys.masked\_columns AS c

JOIN sys.tables AS tbl

ON c.[object\_id] = tbl.[object\_id]

WHERE is\_masked = 1;

## Limitations and restrictions

Users with CONTROL SERVER or CONTROL at the database level could view masked data in its original form. This includes admin users or roles such as sysadmin, db\_owner etc.

A masking rule can't be defined for the following column types:

* Encrypted columns (Always Encrypted)
* FILESTREAM
* COLUMN\_SET or a sparse column that is part of a column set.
* A mask can't be configured on a computed column, but if the computed column depends on a column with a MASK, then the computed column returns masked data.
* A column with data masking can't be a key for a FULLTEXT index.
* A column in a PolyBase [external table](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-table-transact-sql?view=sql-server-ver16).

For users without the **UNMASK** permission, the deprecated **READTEXT**, **UPDATETEXT**, and **WRITETEXT** statements don't function properly on a column configured for Dynamic Data Masking.

Adding a dynamic data mask is implemented as a schema change on the underlying table, and therefore can't be performed on a column with dependencies (for example, column referenced by computed column). Trying to add dynamic data mask against columns with dependency will result in an error, ALTER TABLE ALTER COLUMN \_columnname\_ failed because one or more objects access this column. To work around this restriction, you can first remove the dependency, then add the dynamic data mask and then re-create the dependency. For example, if the dependency is due to an index dependent on that column, you can drop the index, then add the mask, and then re-create the dependent index.

Whenever you project an expression referencing a column for which a data masking function is defined, the expression is also masked. Regardless of the function (default, email, random, custom string) used to mask the referenced column, the resulting expression will always be masked with the default function.

Cross database queries spanning two different Azure SQL databases or databases hosted on different SQL Server Instances, and involve any kind of comparison or join operation on MASKED columns do not provide correct results. The results returned from the remote server are already in MASKED form and not suitable for any kind of comparison or join operation locally.

**Note**

Dynamic data masking is not supported when the underlying base table is referenced in an indexed view.

## Security Note: Bypassing masking using inference or brute-force techniques

Dynamic data masking is designed to simplify application development by limiting data exposure in a set of predefined queries used by the application. While Dynamic Data Masking can also be useful to prevent accidental exposure of sensitive data when accessing a production database directly, it's important to note that unprivileged users with ad hoc query permissions can apply techniques to gain access to the actual data. If there's a need to grant such ad hoc access, Auditing should be used to monitor all database activity and mitigate this scenario.

As an example, consider a database principal that has sufficient privileges to run ad hoc queries on the database, and tries to 'guess' the underlying data and ultimately infer the actual values. Assume that we have a mask defined on the [Employee].[Salary] column, and this user connects directly to the database and starts guessing values, eventually inferring the [Salary] value in the Employees table:

SQLCopy

SELECT ID, Name, Salary FROM Employees

WHERE Salary > 99999 and Salary < 100001;

Expand table

| **Id** | **Name** | **Salary** |
| --- | --- | --- |
| 62543 | Jane Doe | 0 |
| 91245 | John Smith | 0 |

This demonstrates that dynamic data masking shouldn't be used alone to fully secure sensitive data from users running ad hoc queries on the database. It's appropriate for preventing accidental sensitive data exposure, but doesn't protect against malicious intent to infer the underlying data.

It's important to properly manage the permissions on the database, and to always follow the minimal required permissions principle. Also, remember to have Auditing enabled to track all activities taking place on the database.

## Granular permissions introduced in SQL Server 2022

Starting with SQL Server 2022 (16.x), you can prevent unauthorized access to sensitive data and gain control by masking it to an unauthorized user at different levels of the database. You can grant or revoke **UNMASK** permission at the database-level, schema-level, table-level or at the column-level to a user, database role, Microsoft Entra identity or Microsoft Entra group. This enhancement provides a more granular way to control and limit unauthorized access to data stored in the database and improve data security management.

## Examples

### Create a dynamic data mask

The following example creates a table with three different types of dynamic data masks. The example populates the table, and selects to show the result.

SQLCopy

-- schema to contain user tables

CREATE SCHEMA Data;

GO

-- table with masked columns

CREATE TABLE Data.Membership (

MemberID INT IDENTITY(1, 1) NOT NULL PRIMARY KEY CLUSTERED,

FirstName VARCHAR(100) MASKED WITH (FUNCTION = 'partial(1, "xxxxx", 1)') NULL,

LastName VARCHAR(100) NOT NULL,

Phone VARCHAR(12) MASKED WITH (FUNCTION = 'default()') NULL,

Email VARCHAR(100) MASKED WITH (FUNCTION = 'email()') NOT NULL,

DiscountCode SMALLINT MASKED WITH (FUNCTION = 'random(1, 100)') NULL

);

-- inserting sample data

INSERT INTO Data.Membership (FirstName, LastName, Phone, Email, DiscountCode)

VALUES

('Roberto', 'Tamburello', '555.123.4567', 'RTamburello@contoso.com', 10),

('Janice', 'Galvin', '555.123.4568', 'JGalvin@contoso.com.co', 5),

('Shakti', 'Menon', '555.123.4570', 'SMenon@contoso.net', 50),

('Zheng', 'Mu', '555.123.4569', 'ZMu@contoso.net', 40);

GO

A new user is created and granted the **SELECT** permission on the schema where the table resides. Queries executed as the MaskingTestUser view masked data.

SQLCopy

CREATE USER MaskingTestUser WITHOUT LOGIN;

GRANT SELECT ON SCHEMA::Data TO MaskingTestUser;

-- impersonate for testing:

EXECUTE AS USER = 'MaskingTestUser';

SELECT \* FROM Data.Membership;

REVERT;

The result demonstrates the masks by changing the data from:

1 Roberto Tamburello 555.123.4567 RTamburello@contoso.com 10

into:

1 Rxxxxxo Tamburello xxxx RXXX@XXXX.com 91

where the number in DiscountCode is random for every query result.

### Add or editing a mask on an existing column

Use the ALTER TABLE statement to add a mask to an existing column in the table, or to edit the mask on that column.  
The following example adds a masking function to the LastName column:

SQLCopy

ALTER TABLE Data.Membership

ALTER COLUMN LastName ADD MASKED WITH (FUNCTION = 'partial(2,"xxxx",0)');

The following example changes a masking function on the LastName column:

SQLCopy

ALTER TABLE Data.Membership

ALTER COLUMN LastName VARCHAR(100) MASKED WITH (FUNCTION = 'default()');

### Grant permissions to view unmasked data

Granting the **UNMASK** permission allows MaskingTestUser to see the data unmasked.

SQLCopy

GRANT UNMASK TO MaskingTestUser;

EXECUTE AS USER = 'MaskingTestUser';

SELECT \* FROM Data.Membership;

REVERT;

-- Removing the UNMASK permission

REVOKE UNMASK TO MaskingTestUser;

### Drop a dynamic data mask

The following statement drops the mask on the LastName column created in the previous example:

SQLCopy

ALTER TABLE Data.Membership

ALTER COLUMN LastName DROP MASKED;

### Granular permission examples

1. Create schema to contain user tables:

SQLCopy

CREATE SCHEMA Data;

GO

1. Create table with masked columns:

SQLCopy

CREATE TABLE Data.Membership (

MemberID INT IDENTITY(1, 1) NOT NULL PRIMARY KEY CLUSTERED,

FirstName VARCHAR(100) MASKED WITH (FUNCTION = 'partial(1, "xxxxx", 1)') NULL,

LastName VARCHAR(100) NOT NULL,

Phone VARCHAR(12) MASKED WITH (FUNCTION = 'default()') NULL,

Email VARCHAR(100) MASKED WITH (FUNCTION = 'email()') NOT NULL,

DiscountCode SMALLINT MASKED WITH (FUNCTION = 'random(1, 100)') NULL,

BirthDay DATETIME MASKED WITH (FUNCTION = 'default()') NULL

);

1. Insert sample data:

SQLCopy

INSERT INTO Data.Membership (FirstName, LastName, Phone, Email, DiscountCode, BirthDay)

VALUES

('Roberto', 'Tamburello', '555.123.4567', 'RTamburello@contoso.com', 10, '1985-01-25 03:25:05'),

('Janice', 'Galvin', '555.123.4568', 'JGalvin@contoso.com.co', 5, '1990-05-14 11:30:00'),

('Shakti', 'Menon', '555.123.4570', 'SMenon@contoso.net', 50, '2004-02-29 14:20:10'),

('Zheng', 'Mu', '555.123.4569', 'ZMu@contoso.net', 40, '1990-03-01 06:00:00');

1. Create schema to contain service tables:

SQLCopy

CREATE SCHEMA Service;

GO

1. Create service table with masked columns:

SQLCopy

CREATE TABLE Service.Feedback (

MemberID INT IDENTITY(1, 1) NOT NULL PRIMARY KEY CLUSTERED,

Feedback VARCHAR(100) MASKED WITH (FUNCTION = 'default()') NULL,

Rating INT MASKED WITH (FUNCTION = 'default()'),

Received\_On DATETIME

);

1. Insert sample data:

SQLCopy

INSERT INTO Service.Feedback(Feedback, Rating, Received\_On)

VALUES

('Good', 4, '2022-01-25 11:25:05'),

('Excellent', 5, '2021-12-22 08:10:07'),

('Average', 3, '2021-09-15 09:00:00');

1. Create different users in the database:

SQLCopy

CREATE USER ServiceAttendant WITHOUT LOGIN;

GO

CREATE USER ServiceLead WITHOUT LOGIN;

GO

CREATE USER ServiceManager WITHOUT LOGIN;

GO

CREATE USER ServiceHead WITHOUT LOGIN;

GO

1. Grant read permissions to the users in the database:

SQLCopy

ALTER ROLE db\_datareader ADD MEMBER ServiceAttendant;

ALTER ROLE db\_datareader ADD MEMBER ServiceLead;

ALTER ROLE db\_datareader ADD MEMBER ServiceManager;

ALTER ROLE db\_datareader ADD MEMBER ServiceHead;

1. Grant different UNMASK permissions to users:

SQLCopy

--Grant column level UNMASK permission to ServiceAttendant

GRANT UNMASK ON Data.Membership(FirstName) TO ServiceAttendant;

-- Grant table level UNMASK permission to ServiceLead

GRANT UNMASK ON Data.Membership TO ServiceLead;

-- Grant schema level UNMASK permission to ServiceManager

GRANT UNMASK ON SCHEMA::Data TO ServiceManager;

GRANT UNMASK ON SCHEMA::Service TO ServiceManager;

--Grant database level UNMASK permission to ServiceHead;

GRANT UNMASK TO ServiceHead;

1. Query the data under the context of user ServiceAttendant:

SQLCopy

EXECUTE AS USER = 'ServiceAttendant';

SELECT MemberID, FirstName, LastName, Phone, Email, BirthDay

FROM Data.Membership;

SELECT MemberID, Feedback, Rating

FROM Service.Feedback;

REVERT;

1. Query the data under the context of user ServiceLead:

SQLCopy

EXECUTE AS USER = 'ServiceLead';

SELECT MemberID, FirstName, LastName, Phone, Email, BirthDay

FROM Data.Membership;

SELECT MemberID, Feedback, Rating

FROM Service.Feedback;

REVERT;

1. Query the data under the context of user ServiceManager:

SQLCopy

EXECUTE AS USER = 'ServiceManager';

SELECT MemberID, FirstName, LastName, Phone, Email, BirthDay

FROM Data.Membership;

SELECT MemberID, Feedback, Rating

FROM Service.Feedback;

REVERT;

1. Query the data under the context of user ServiceHead

SQLCopy

EXECUTE AS USER = 'ServiceHead';

SELECT MemberID, FirstName, LastName, Phone, Email, BirthDay

FROM Data.Membership;

SELECT MemberID, Feedback, Rating

FROM Service.Feedback;

REVERT;

1. To revoke UNMASK permissions, use the following T-SQL statements:

SQLCopy

REVOKE UNMASK ON Data.Membership(FirstName) FROM ServiceAttendant;

REVOKE UNMASK ON Data.Membership FROM ServiceLead;

REVOKE UNMASK ON SCHEMA::Data FROM ServiceManager;

REVOKE UNMASK ON SCHEMA::Service FROM ServiceManager;

REVOKE UNMASK FROM ServiceHead;

## Related content

* [CREATE TABLE (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16)
* [ALTER TABLE (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver16)
* [column\_definition (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-column-definition-transact-sql?view=sql-server-ver16)
* [sys.masked\_columns (Transact-SQL)](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-masked-columns-transact-sql?view=sql-server-ver16)
* [Get started with SQL Database Dynamic Data Masking (Azure portal)](https://learn.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview)
* [Dynamic data masking in Fabric data warehousing](https://learn.microsoft.com/en-us/fabric/data-warehouse/dynamic-data-masking)