SAP HANA Security Guide for SAP HANA Platform

Generated on: 2024-11-25 09:53:27 GMT+0000

SAP HANA Platform | 2.0 SPS 08

PUBLIC

Original content: https://help.sap.com/docs/SAP_HANA_PLATFORM/b3ee5778bc2e4a089d3299b82ec762a7?locale=en-US&state=PRODUCTION&version=2.0.08

9.12

SAP HANA Data Masking

9.12

Data masking provides an additional layer of access control that can be applied to tables and views.

A column mask protects sensitive or confidential data in a particular column of a table or view by transforming the data in such a way that it is only visible partially or rendered completely meaningless for an unprivileged user, while still appearing real and consistent.

i Note

Data masking scrambles data during execution only. Data is not encrypted at the physical (disk) layer.

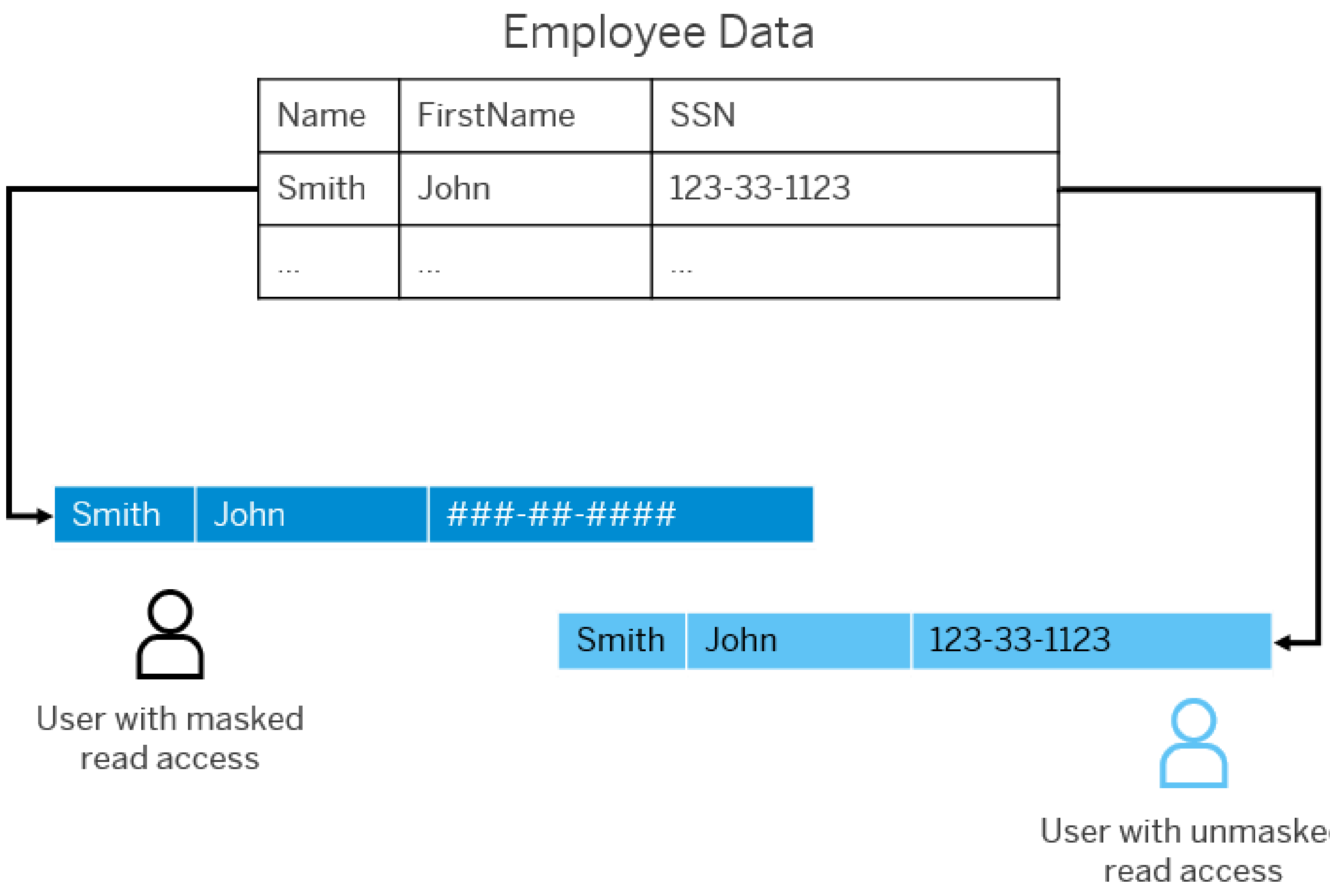
9.12

Unlike the all-or-nothing approach of authorization in which an unprivileged user gets a "not authorized" error, a user who is not privileged to see masked data does not get an error, but instead sees only the masked data.

i Note

A user still needs the SELECT privilege on a table or view with data masking. Only a user who also has the UNMASKED object privilege for the table or view is able to see the original data.

Masking is useful in situations where strict data protection regulations apply and a second layer of protection is required on top of authorization to safeguard specific data. For example, a highly-privileged administrator has access to company employee data, but instead of seeing the actual content of the column containing social security numbers, he sees masked values (for example, ###-##-####).



Supported Objects

You can specify a mask for columns in rowstore and columnstore tables, as well as in SQL and calculations views.

Masking supports the following string data types:

- VARCHAR
- NVARCHAR
- SHORTEXT
- ALPHANUM

It is also possible to nest views containing masked columns.

Unsupported Objects

Adding mask expressions for the following types of views or tables and/or columns results in a `feature not supported` error:

- Unsupported view types:
    - Join and OLAP views
    - Hierarchy views
    - Parametrized SQL views
    - Transient column views
- Unsupported table types:
    - Temporary tables
    - Extended storage (Dynamic Tiering) and proxy tables
    - Replica tables
    - Virtual tables
    - Document store (DocStore) tables
- Unsupported column types:
    - Columns encrypted using with client-side encryption
    - Calculated and constant columns
- Unsupported object hierarchies:
    - Join and OLAP views cannot be defined on top of tables with masked columns

Implementation Considerations

When masking is used, correlations between masked and unmasked data may allow a user to infer the masked value. For example, when table partitioning is used, and if a masked column is used for range partitioning, a user with the SELECT object privilege on a table or view may be able to use the row ID to determine in which partition a particular row is located.

Related Information

SAP HANA Data Anonymization

Masking Definition

Masking is defined as part of the table or view. The mask mode specifies how access to masked data is determined in object hierarchies, and the mask expression specifies how the data in a column should be returned to unauthorized users.

Column masking is defined directly in the table or view using the CREATE TABLE and CREATE VIEW statements, for example:

```
CREATE VIEW <view_name> (<column_name_list>) AS <subquery>
WITH [DEFAULT | SESSION USER]
MASK (<column_name> USING '<mask_expression>');
```

For column masking, you define:

- **Mask mode**

  When data in masked columns is accessed through **object hierarchies**, the mask mode specifies how the user accessing the data is determined for the authorization check. There are two mask modes: DEFAULT and SESSION USER.

  With DEFAULT mask mode, the privileges of the owner of the next higher-level object with definer security mode are checked. With SESSION USER mask mode, the session user is always subject to the authorization check.

  If you do not set a mask mode, DEFAULT mode is automatically applied.

  For more detailed information about how these mask modes impact the authorization process, see *Authorization in Masked Tables and Views*.

- **Mask expression**

  The mask expression specifies how the data in a specific column should be returned to unauthorized users. It can be specified using any SQL expression that SAP HANA supports, such as SQL functions or user-defined functions. However, it must return data of the same type and length as the original data. This ensures that masked data appears real and consistent.

For example, a new SQL table with DEFAULT mask mode and column mask as a constant expression can be created as follows:

```
CREATE TABLE T(FirstName, Name, MaskedSSN, MaskedCreditcard)
WITH DEFAULT
MASK (MaskedSSN USING 'XXXX-XX-' || RIGHT(MaskedSSN, 4), MaskedCreditcard USING 'XXXX-XXXX-XXXX-XX
```

A view with SESSION USER mask mode can be created as follows:

```
CREATE VIEW V(FirstName, Name, MaskedSSN, MaskedCreditcard) AS SELECT FirstName, Name, SSN, Credit
WITH SESSION USER
MASK (MaskedSSN USING 'XXXX-XX-' || RIGHT(MaskedSSN, 4), MaskedCreditcard USING 'XXXX-XXXX-XXXX-XX
```

For more information about the syntax of the MASK clause, see the relevant statements in the *SAP HANA SQL and Systems View Reference*.

i Note

Table or view creation may fail with the error message below if SAP HANA cannot calculate the length of the resulting mask expression or if it exceeds the column length:

`SAP DBTech JDBC: [5735]: Masking: invalid mask expression: exception 4075006: mask expression <masking expression> returns a value potentially too long for the column <column_name>`

In this case, limit the mask expression using other string functions such as:

- SUBSTRING (*<mask_expression>*, 0, *<column_length>*)
- CAST (*<mask_expression>* as *<column_data_type>*)

i Note

You can add masked columns to existing tables and views (for example, to apply masking in an existing application), as well as modify and delete masking definitions using the ALTER TABLE and ALTER VIEW statements. You can add a mask to a column in a calculation view using the SAP Web IDE for SAP HANA.

Related Information

CREATE VIEW Statement (Data Definition)
CREATE TABLE Statement (Data Definition)
ALTER TABLE Statement (Data Definition)
ALTER VIEW Statement (Data Definition)
Mask Column Values in Client Tools (SAP HANA Modeling Guide for SAP Web IDE for SAP HANA)

Authorization in Masked Tables and Views

The object privilege `UNMASKED` controls the visibility of unmasked data. In object hierarchies, the mask mode of the object with masked columns determines whose authorization must be checked.

Access to tables or views containing masked columns is controlled by the usual object privileges, for example, `SELECT` for read access. Data in masked columns is returned in line with the user's privileges, but it is always masked, regardless of query type.

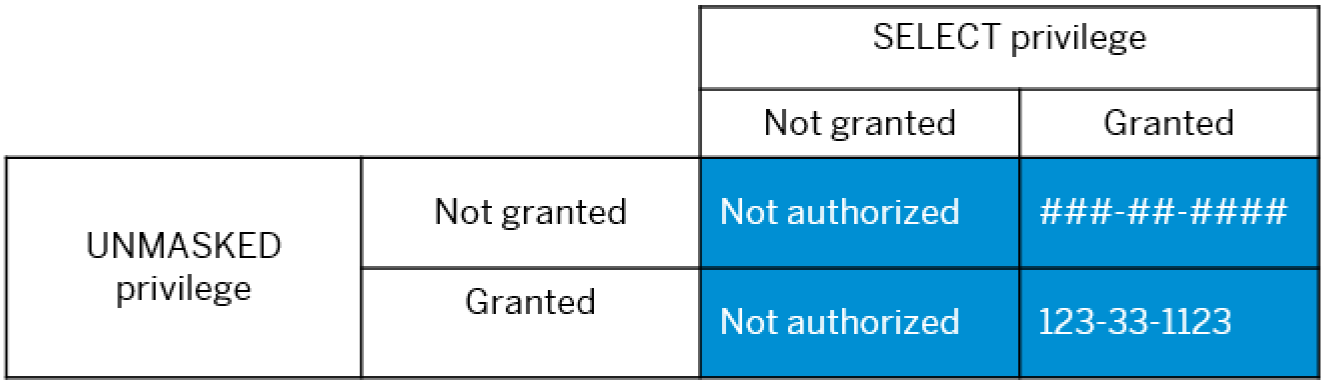
The ability to see the real, unmasked data is controlled by the object privilege `UNMASKED`.

i Note

Not having the `UNMASKED` privilege does not result in an authorization error, assuming the user is otherwise sufficiently authorized.

As an object privilege, `UNMASKED` can initially be granted only by the object owner or the owner of the schema that contains the object. For more information, see the section on object privileges.

The following table illustrates how different combinations of the `SELECT` and `UNMASKED` privileges control what a user sees on simple read access to a masked object:



		SELECT privilege	
		Not granted	Granted
UNMASKED privilege	Not granted	Not authorized	###-##-####
	Granted	Not authorized	123-33-1123

Authorization for Simple Read Access

For access to masked data in object hierarchies, the mask mode of the object with masked columns and the authorization of the accessing user determine visibility of unmasked data.

Authorization with DEFAULT Mask Mode

If a masked view or table is defined with DEFAULT mask mode, the SQL security mode (definer or invoker) of higher-level objects determines the user who is actually accessing the masked columns.

About the SQL Security Mode of Complex Objects

Complex SQL objects, such as procedures, functions, or views, can be defined on top of other objects in either "definer" or "invoker" security mode. The security mode determines how the authorization to the underlying objects is checked. A **definer object** uses the privileges of its owner to access the underlying objects. By contrast, an **invoker object** simply provides access through to its underlying objects and expects the querying user to provide all required privileges on the underlying objects.

Examples of definer and invoker objects:

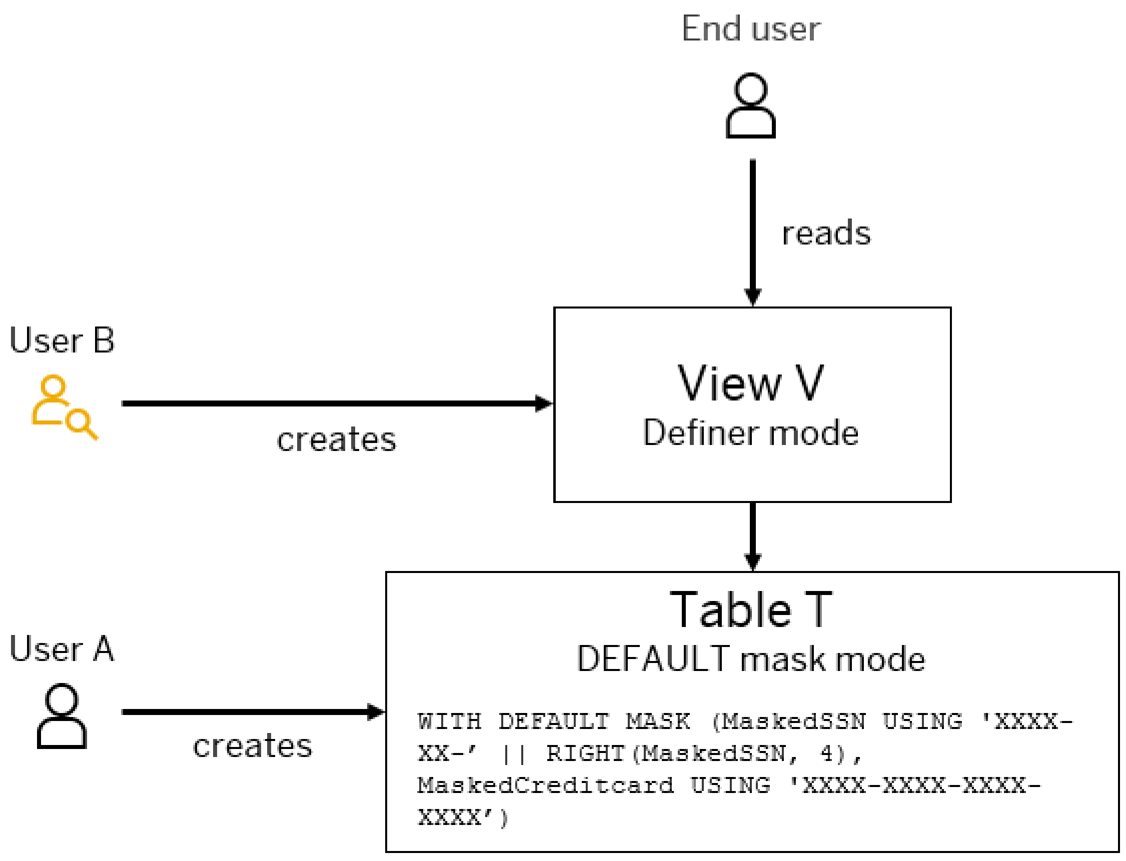
- **Row views** (in other words, views created with the `CREATE VIEW` SQL statement) are considered definer objects as the owner is used to access the underlying objects.
- **SQLScript procedures** can be defined with either definer or invoker security mode.
- **User-defined functions** can be defined with either definer or invoker security mode.

For the syntax of the statements to create these object types, see the *SAP HANA SQL Reference*.

Masking with Definer-Dependent Objects

The owner of a definer object is the user who accesses the underlying objects. This means that if an underlying object contains columns masked using the DEFAULT mask mode, the owner of the definer object is the user who is checked for the UNMASKED privilege. This can be illustrated with the following example.

User A creates a table T with masked columns using the DEFAULT mode, and user B creates a view V on top of T. The end user has read access on V, in other words: the end user has SELECT privilege on V and user B has SELECT WITH GRANT OPTION on T.


End user
reads
User B
creates
View V
Definer mode
Table T
DEFAULT mask mode
User A
creates
WITH DEFAULT MASK (MaskedSSN USING 'XXXX-XX-' || RIGHT(MaskedSSN, 4), MaskedCreditcard USING 'XXXX-XXXX-XXXX-XXXX')

Authorization for Object Hierarchy with Definer Dependencies

Whether the end user sees masked or unmasked data from T depends entirely on user B as the **owner of the definer object**:

- If user B does not have the UNMASKED privilege on T, data from T is masked with the T mask expression, processed within V and then returned to the end-user.
- If user B does have the UNMASKED privilege on T, unmasked data from T is processed in V and returned to the end-user.

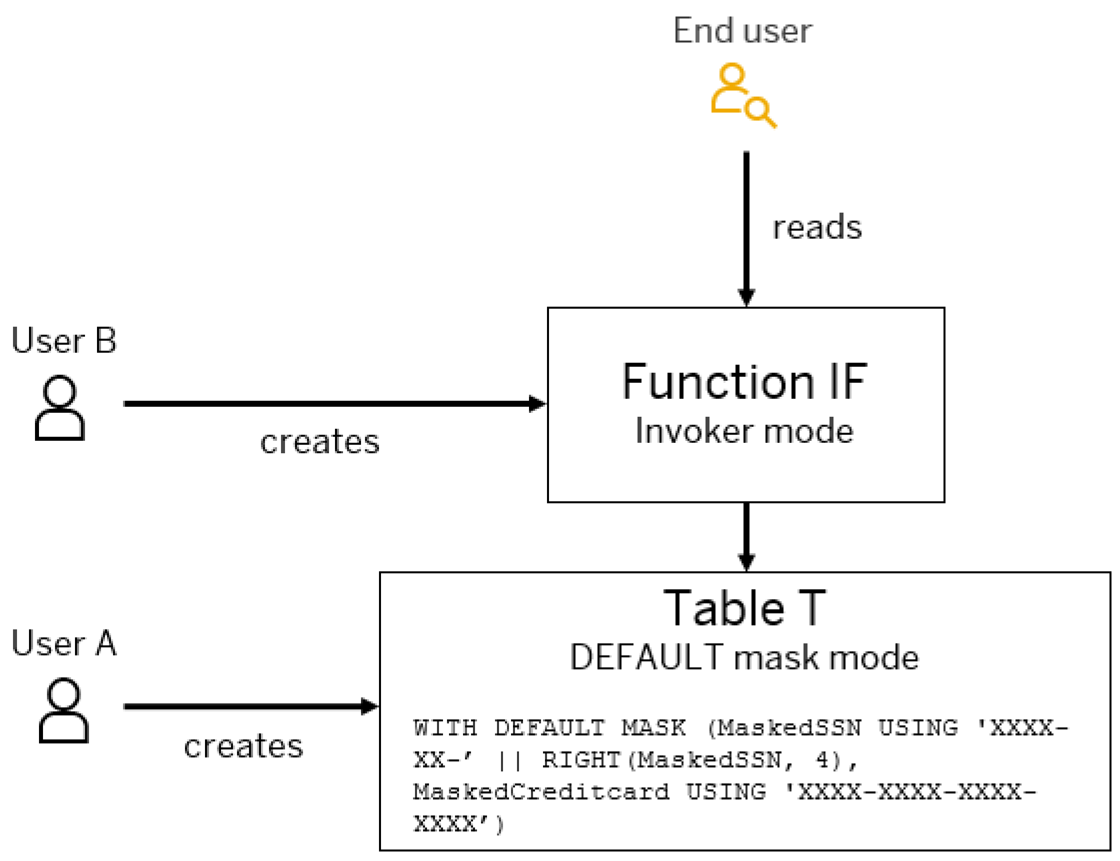
Note

All views, regardless of type, are considered definer objects for the purposes of determining authorization on masked columns. This means that view owners will be checked for the UNMASKED privilege as shown in the example above if they are built on top of tables or other views with DEFAULT mask mode.

Masking with Invoker Dependent Objects

Invoker objects simply provide access through to the underlying objects. This means that if an underlying object contains columns masked using the DEFAULT mask mode, the owner of the invoker object is not considered. Instead, the owner of the next higher-level definer object, if found, or the end user is checked for the UNMASKED privilege. This can be illustrated with the following example.

User A creates a table T with masked columns using DEFAULT masking mode, and user B creates an invoker function IF on top of T. In order to successfully access IF, the end user needs not only EXECUTE privilege on IF, but also as SELECT privilege on T since IF is an invoker object.


End user
reads
User B
creates
Function IF
Invoker mode
User A
creates
Table T
DEFAULT mask mode
WITH DEFAULT MASK (MaskedSSN USING 'XXXX-XX-' || RIGHT(MaskedSSN, 4), MaskedCreditcard USING 'XXXX-XXXX-XXXX-XXXX')

Authorization for Object Hierarchy with Invoker Dependencies

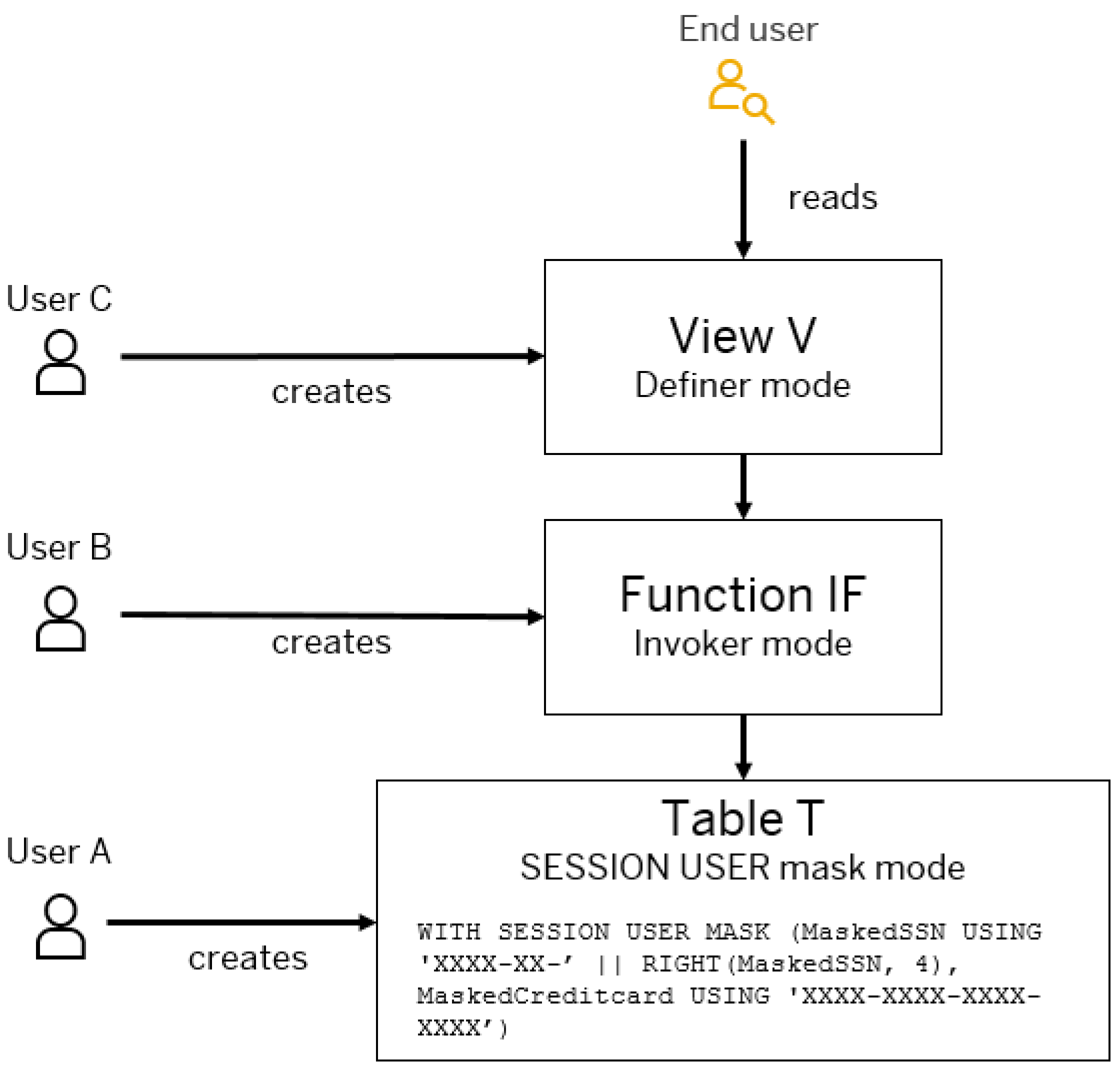
How data in the masked column in T is processed within IF and returned to the end user depends on the whether or not the **end user** has the UNMASKED privilege on T as follows:

- If the end user does not have the UNMASKED privilege on T, data from T is masked with the T mask expression, processed within IF and then returned to the end-user.
- If the end user does have the UNMASKED privilege on T, full data from T is processed in IF and returned to the end-user.

SESSION USER Mask Mode

If a masked view is defined with the SESSION USER mask mode, the session user is always considered the user who accesses the masked columns. This is the user who is checked for the UNMASKED privilege regardless of the security mode of higher-level objects built on top of the object with masked columns. This can be illustrated with the following example.

User A creates a table T with masked columns using SESSION USER masking mode, user B creates an invoker function IF on top of T, and user C creates a definer view V on top of IF. The end user is already able to select from V.



Authorization for Object Hierarchy with SESSION USER Mask Mode

Due to the fact that T uses SESSION USER mask mode, how data in the masked column in T is obtained and further processed by IF and V depends on whether or not the end user has the UNMASKED privilege on T as follows.

- If the end user does not have the UNMASKED privilege on T, data from T is masked with the T mask expression, processed within IF and then returned to the end-user.
- If the end user does have the UNMASKED privilege on T, full data from T is processed in IF and returned to the end-user.

Implementation Considerations

Mask Mode Impact

The mask mode does not always affect the data returned to the end user. In the following situations, the result will be the same regardless of the mask mode:

- Masking is applied to the top-level object in an object hierarchy

In this case, regardless of the mask mode or security mode of the top-level object, the users accessing the top-level object are checked for the UNMASKED privilege.

- Only procedures and user-defined functions with invoker mode are built on top of an object with masked columns

  The end users accessing any objects in such an object hierarchy are checked for the UNMASKED privilege.

Combining Masking and Analytic Privileges

- Combinations within a single object

  It is not possible to use different mask modes within a single table or a view.

  It is possible to create a view containing masked columns that is also secured with analytic privileges. In this case, the end user must have an applicable analytic privilege in addition to the SELECT privilege. If this is the case, the authorization check first filters the data based on the conditions specified in the analytic privilege, and then masks any data in masked columns. This means that a user who is authorized to see unmasked data still only sees the data that he or she is authorized to see as specified in the analytic privilege.

- Combinations within an object hierarchy

  It is possible to build objects with different mask modes on top of each other in deeply nested hierarchies. However, it is important to validate the results for unauthorized users carefully. While it is not a problem for users with authorization on the complete object hierarchy, different push-down decisions of query optimization might result in different results with masked data for unauthorized users. In particular, if aggregation is executed on top of masked data, wrong results could be returned unnoticed as no error occurs.

  When building analytic privileges on top of masked data, please keep in mind that the way in which you define your mask might influence whether or not a row is filtered in your analytic privilege.

Related Information

CREATE TABLE Statement (Data Definition)
CREATE VIEW Statement (Data Definition)
CREATE PROCEDURE Statement (Procedural)
CREATE FUNCTION Statement (Procedural)
Analytic Privileges
Object Privileges

Analysis of Effective Mask Expressions

The system view SYS.EFFECTIVE_MASK_EXPRESSIONS can be used to analyze which mask expressions are currently being applied, for example, if a query on an object is returning unexpected masked data. An administration user with the CATALOG READ system privilege can analyze unexpected query results for other users.

The system view EFFECTIVE_MASK_EXPRESSIONS resolves the complete object hierarchy for a given object and displays the following information for each level of dependency, that is, each pair of dependent object and underlying object:

- The mask expression if the underlying object defines a mask expression of the column of interest
- The actual user accessing the underlying object according to the mask mode of the underlying object and security mode of the dependent object
- If the mask expression is being enforced depending on whether or not the actual accessing user has the UNMASKED privilege on the object with masked columns

Note

Information on masking can also be traced for analysis and debugging purposes using trace component `DATAMASKING` level `INFO` for the `indexserver.ini`.

Related Information

EFFECTIVE_MASK_EXPRESSIONS System View

Example: Masking Data with DEFAULT Mask Mode

This example shows how to mask data in a view with DEFAULT mask mode.

Procedure

1. User `data_owner` creates a table (`credit_tab`) containing credit card data:

```
CREATE TABLE credit_tab (Name varchar(20), CREDIT_CARD varchar(19));
INSERT INTO credit_tab values ('John', '1111-1111-1111-1111');
INSERT INTO credit_tab values ('James', '2222-2222-2222-2222');
```

2. User `data_owner` creates a view on the credit card table (`credit_view`), masking the credit card number column and specifying DEFAULT mask mode:

```
CREATE VIEW credit_view AS SELECT * FROM credit_tab
WITH DEFAULT MASK (CREDIT_CARD USING LEFT(CREDIT_CARD,4) || '-XXXX-XXXX-' || RIGHT(CREDIT_CAR
```

→ Tip

To mask the same data in different views, you can create a reusable SQL function to generate the mask expression. This is shown in *Example: Masking Data in View Hierarchy with Structured Privilege Check*.

3. User `data_owner` grants user `end_user` access to the view:

```
GRANT SELECT ON credit_view TO end_user;
```

Results

When user `end_user` selects from the credit card view, the data is masked as follows:

```
 ;NAME ;CREDIT_CARD
1;John ;1111-XXXX-XXXX-1111
2;James;2222-XXXX-XXXX-2222
```

Example: Masking Data in Object Hierarchy with SESSION USER Mask Mode

This example shows the behavior of a user-defined invoker function built on top of a view with a masked column.

Procedure

1. User `data_owner1` creates the credit card table (`credit_tab`), masking the credit card number column:

```
CREATE TABLE credit_tab (Name varchar(20), CREDIT_CARD varchar(19));
INSERT INTO credit_tab values ('John', '1111-1111-1111-1111');
INSERT INTO credit_tab values ('James', '2222-2222-2222-2222');
ALTER TABLE CREDIT_TAB ADD SESSION USER MASK (CREDIT_CARD USING LEFT(CREDIT_CARD,4) || '-XXXX
```

→ **Tip**

To mask the same data in different views, you can create a reusable SQL function to generate the mask expression. This is shown in *Example: Masking Data in View Hierarchy with Structured Privilege Check*.

2. User `data_owner1` grants select authorization on `credit_tab` to user `data_owner2`:

```
GRANT SELECT ON credit_tab TO data_owner2 WITH GRANT OPTION;
```

3. User `data_owner2` creates a table function on `credit_tab` with definer security mode and a SQL view on top of the function:

```
CREATE FUNCTION credit_function
RETURNS TABLE(Name varchar(20), CREDIT_CARD varchar(19))
LANGUAGE SQLSCRIPT SQL SECURITY DEFINER
AS
BEGIN
        RETURN SELECT * FROM data_owner1.credit_tab;
END;
CREATE VIEW credit_view AS SELECT * FROM credit_function();
```

4. User `data_owner2` grants select authorization to user `end_user`:

```
GRANT SELECT ON credit_view TO end_user
```

Results

When user `end_user` selects from the `credit_view`, the data is masked according the mask expression defined in `data_owner1.credit_tab`.

User `end_user` can see full (unmasked) data only after `data_owner1` grants the UNMASKED privilege on the table `credit_tab`.

Masking Explanation with SYS.EFFECTIVE_MASK_EXPRESSIONS

The effectively enforced mask expressions can be analyzed with the help of the system view SYS.EFFECTIVE_MASK_EXPRESSION. Using a user with the system privilege CATALOG READ, the system view can be queried for our scenario as follows:

```
SELECT
        EFFECTIVE_USER_NAME,
        DEPENDENT_OBJECT_NAME,
        UNDERLYING_OBJECT_NAME,
        UNDERLYING_COLUMN_NAME,
        MASK_EXPRESSION,
        MASK_EXPRESSION_STATUS
FROM SYS.EFFECTIVE_MASK_EXPRESSIONS
WHERE USER_NAME = 'END_USER'
        AND ROOT_SCHEMA_NAME = 'DATA_OWNER2'
        AND ROOT_OBJECT_NAME = 'CREDIT_VIEW'
        AND ROOT_COLUMN_NAME = 'CREDIT_CARD';
```

In first case above, the user `end_user` does not have UNMASKED privilege on the table `credit_tab`, so we see the following result:

Output Code

```
  ;EFFECTIVE_USER_NAME    ;DEPENDENT_OBJECT_NAME   ;UNDERLYING_OBJECT_NAME           ;UNDERLYING_COLU
 1;END_USER               ;                        ;CREDIT_VIEW                      ;CREDIT_CARD
 2;DATA_OWNER2            ;CREDIT_VIEW             ;CREDIT_FUNCTION                  ;
 3;END_USER               ;CREDIT_FUNCTION         ;CREDIT_TAB                       ;CREDIT_CARD
```

In particular, due to the setting of SESSION USER mask mode in `credit_tab`, the effective user is end_user. This is the user who needs the UNMASKED privilege on the table in order to see full data, otherwise the masked expression shown in the column MASK_EXPRESSION is enforced as indicated by the column MASK_EXPRESSION_STATUS.

If end_user is granted the UNMASKED privilege on the credit_tab, we see the following result. The MASK_EXPRESSION_STATUS column indicates that the mask expression is not being enforced.

Output Code

```
  ;EFFECTIVE_USER_NAME    ;DEPENDENT_OBJECT_NAME   ;UNDERLYING_OBJECT_NAME           ;UNDERLYING_COLU
 1;END_USER               ;                        ;CREDIT_VIEW                      ;CREDIT_CARD
 2;DATA_OWNER2            ;CREDIT_VIEW             ;CREDIT_FUNCTION                  ;
 3;END_USER               ;CREDIT_FUNCTION         ;CREDIT_TAB                       ;CREDIT_CARD
```

Example: Masking Data in a View with Structured Privilege Check

This example shows the behavior of a view that contains a masked column and that is also secured with an analytic privilege.

Prerequisites

User `data_owner` has the system privilege STRUCTUREDPRIVILEGE ADMIN.

Procedure

1. User `data_owner` creates a table (`credit_tab`) containing credit card data:

```
CREATE TABLE credit_tab (Name varchar(20), CREDIT_CARD varchar(19));
INSERT INTO credit_tab values ('John', '1111-1111-1111-1111');
INSERT INTO credit_tab values ('James', '2222-2222-2222-2222');
```

2. User `data_owner` creates a view (`credit_view`) on the credit card table, masking the credit card number column and further securing the view with a structured privilege check:

```
CREATE VIEW credit_view AS SELECT * FROM credit_tab
WITH DEFAULT MASK (CREDIT_CARD USING LEFT(CREDIT_CARD,4) || '-XXXX-XXXX-' || RIGHT(CREDIT_CAR
        STRUCTURED PRIVILEGE CHECK;
```

Tip

To mask the same data in different views, you can create a reusable SQL function to generate the mask expression. This is shown in *Example: Masking Data in View Hierarchy with Structured Privilege Check*.

3. User `data_owner` grants select authorization to user `end_user`:

```
GRANT SELECT ON credit_view TO end_user;
```

4. User `mask_owner` creates an analytic privilege for `credit_view` that allows access to the row containing the credit number 1111-1111-1111-1111:

```
CREATE STRUCTURED PRIVILEGE credit_ap for SELECT ON data_owner.credit_view WHERE CREDIT_CARD
```

5. User `mask_owner` grants the analytic privilege `credit_ap` to user `end_user`:

    ```
    GRANT STRUCTURED PRIVILEGE credit_ap TO end_user;
    ```

Results

When user `end_user` selects from the credit card view, the data is first filtered in line with the analytic privilege and then masked:

```
 ;NAME ;CREDIT_CARD
1;John ;1111-XXXX-XXXX-1111
```

Example: Masking Data in View Hierarchy with Structured Privilege Check

This example shows the behavior of a view secured with an analytic privilege built on top of a view with a masked column.

Prerequisites

User `mask_owner` has the system privilege STRUCTUREDPRIVILEGE ADMIN.

Procedure

1. User `data_owner1` creates a table (`credit_tab`) containing credit card data:

    ```
    CREATE TABLE credit_tab (Name varchar(20), CREDIT_CARD varchar(19));
    INSERT INTO credit_tab values ('John', '1111-1111-1111-1111');
    INSERT INTO credit_tab values ('James', '2222-2222-2222-2222');
    ```

2. User `mask_owner` defines a mask as a SQL function and grants user `data_owner1` authorization to execute:

    ```
    CREATE FUNCTION credit_mask(input varchar(19)) RETURNS output VARCHAR(19) LANGUAGE SQLSCRIPT
    AS
    BEGIN
            output = LEFT(:input,4) || '-XXXX-XXXX-' || RIGHT(:input,4);
    END;

    GRANT EXECUTE ON credit_mask TO data_owner1;
    ```

    i Note

    For illustration purposes, this example uses an SQL function to generate the mask expression. For performance reasons, define simple mask expression directly as part of the table or view definition if possible.

3. User `data_owner1` creates a view on the credit card table (`credit_view_base`), masking the credit card number column using the SQL function:

    ```
    CREATE VIEW credit_view_base AS SELECT * FROM credit_tab
    WITH MASK (CREDIT_CARD USING mask_owner.credit_mask(credit_card));
    ```

4. User `data_owner1` grants select authorization on `credit_view_base` to user `data_owner2`:

    ```
    GRANT SELECT ON credit_view_base TO data_owner2;
    ```

5. User `data_owner2` creates a top-level view on the base view, securing it with a structured privilege check:

    ```
    CREATE VIEW credit_view_top AS SELECT * from data_owner1.credit_view_base
    ```

```
WITH STRUCTURED PRIVILEGE CHECK;
```

> **i Note**
>
> User `data_owner2` does not have the UNMASKED privilege on `credit_view_base`.

6. User `data_owner2` grants select authorization to user `end_user`:

   ```
   GRANT SELECT ON credit_view_top TO end_user;
   ```

7. User `mask_owner` creates an analytic privilege for the top-level view `credit_view_top` that allows access to the row containing the credit number 1111-1111-1111-1111:

   ```
   CREATE STRUCTURED PRIVILEGE credit_ap for SELECT ON data_owner.credit_view_top WHERE CREDIT_C
   ```

8. User `mask_owner` grants the analytic privilege `credit_ap` to user `end_user`:

   ```
   GRANT STRUCTURED PRIVILEGE credit_ap TO end_user;
   ```

Results

When user `end_user` selects from the credit card view, the data is masked first and then filtered in line with the analytic privilege. This returns a empty result set.

SAP HANA Data Anonymization

Anonymization methods available in SAP HANA allow you to gain statistically valid insights from your data while protecting the privacy of individuals.

Why Anonymize?

In a data-driven world, a growing amount of business data contains personal or sensitive information. If this data is to be used by applications for statistical analysis, it must be protected to ensure privacy. Trivial modifications to the data like replacing information that directly identifies an individual such as name or social security number (pseudonymization) or simply removing the information is not enough. Re-identification is still possible, for example if additional information is obtained (referred to as a linkage attack).

Unlike masking and pseudonymization, anonymization methods (also called privacy-enhancing methods) provide a more structured approach to modifying data for privacy protection. The quality of such anonymized or privacy-enhanced data is still sufficient for meaningful analysis. Several anonymization methods exist.

SAP HANA supports the methods *k*-anonymity, *l*-diversity, and differential privacy. Which method provides the most appropriate level of privacy depends on your data and the potential attack scenarios and attackers.

Anonymizing Data in SAP HANA

Data anonymization can be applied to SQL views or calculation views, thus enabling analytics on data while still protecting the privacy of individuals.

You define anonymization views using the SQL CREATE VIEW statement and a WITH ANONYMIZATION clause which specifies an anonymization method and includes corresponding parameters for each column in the view to meet the required privacy level. There are three types of parameter:

- Static parameters - used to configure the view as a whole, such as defining the method ('algorithm') to use

- Column parameters - used to apply configuration values to individual columns of the view
- Hierarchy parameters - used for columns which are 'quasi identifiers' to create generalization hierarchies which determine how specific values can be replaced by more generic values.

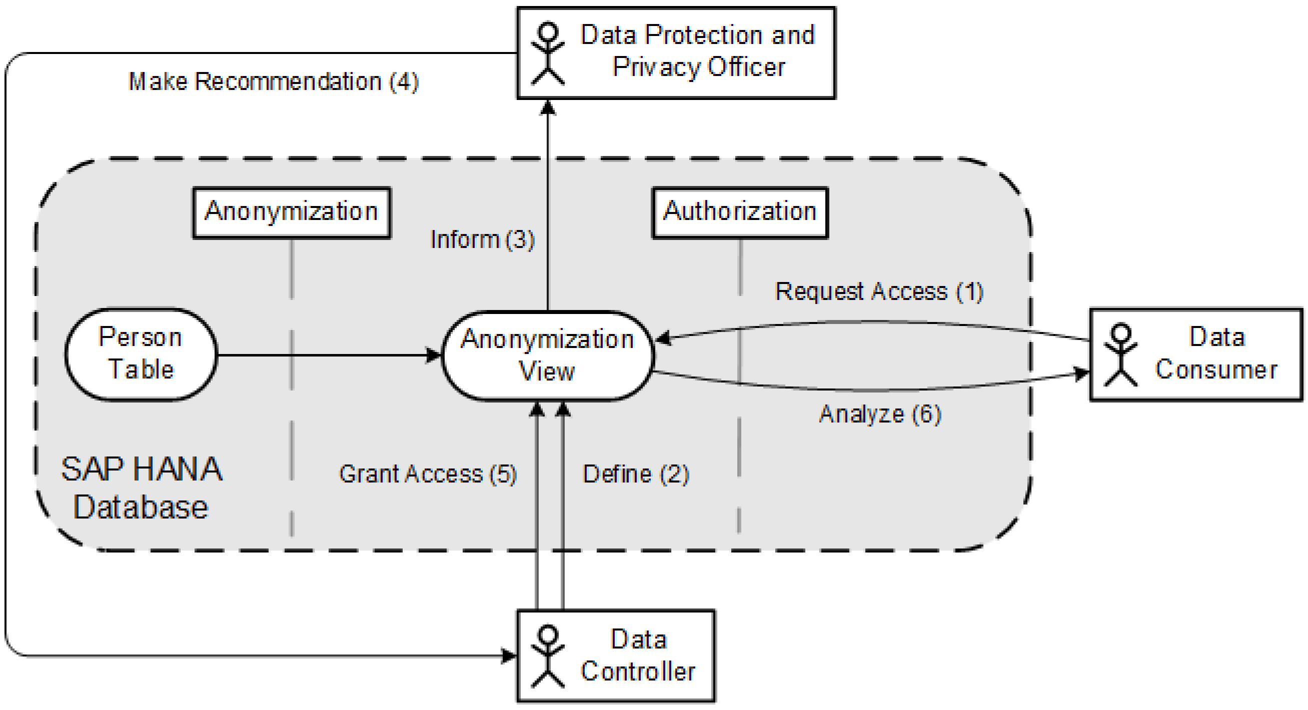
The following example shows an anonymization clause including configuration properties for three columns of a table (ID, GENDER, LOCATION):

```
WITH ANONYMIZATION ( ALGORITHM 'K-ANONYMITY'
PARAMETERS '{"k": 5}'
COLUMN ID PARAMETERS '{"is_sequence": true}'
COLUMN GENDER PARAMETERS '{"is_quasi_identifier":true, "hierarchy":{"embedded": [["F"], ["M"]]}}'
COLUMN LOCATION PARAMETERS '{"is_quasi_identifier":true, "hierarchy":{"embedded": [["Paris", "Fran
```

Note that two of the columns are quasi identifiers and include the required hierarchy parameters to define appropriate generalization hierarchies.

Anonymization Workflow

A data controller – that is someone who determines when and how personal data is accessed and processed – defines an SQL view or a calculation view and configures the parameters of the chosen anonymization method to meet the required privacy level. Access to the anonymized view can then be granted to users using standard SAP HANA authorization mechanisms. An overview of this process is shown below:



Information about anonymized SQL views can be obtained by using the system views ANONYMIZATION_VIEWS, ANONYMIZATION_VIEW_COLUMNS, and M_ANONYMIZATION_VIEWS. For more information about anonymizing data using SQL, see the *SAP HANA SQL Reference Guide*.

A list of all anonymized views as well as all calculation views that have one or more anonymization nodes configured is available in the SAP HANA cockpit (for documentation purposes, for example). For more information about anonymizing data using calculation views, see the *SAP HANA Modeling Guide for SAP Web IDE for SAP HANA* .

Related Information

SAP HANA Data Anonymization Guide

SAP HANA Data Masking