

1.0.0

# Securing Apache Impala

Date published: 2020-11-30

Date modified: 2025-10-22

# CLOUDERA

<https://docs.cloudera.com/>

# Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

- Impala Authorization.....4**
- Row-level filtering in Impala with Ranger policies..... 11**
- AES encryption and decryption support.....11**
  - Encryption and Decryption Examples..... 12

# Impala Authorization

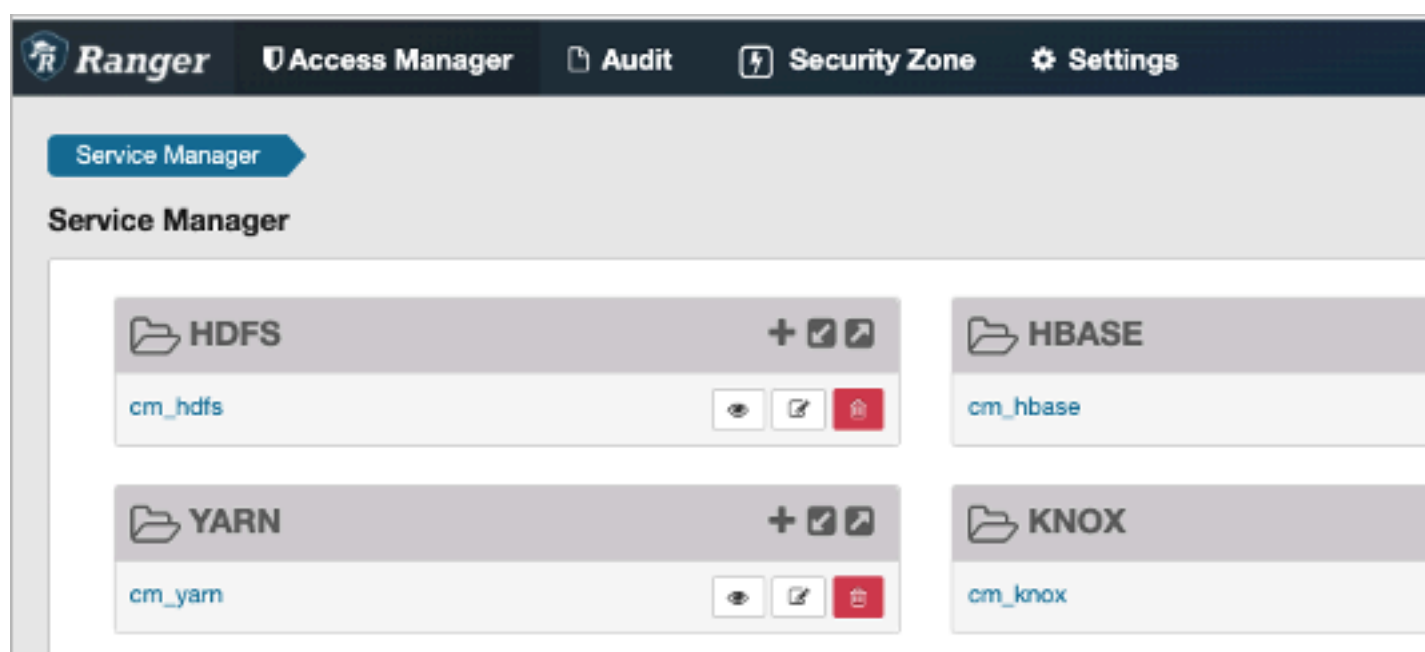
Authorization determines which users are allowed to access which resources, and what operations they are allowed to perform. You use Apache Ranger to enable and manage authorization in Impala.

## Supported Ranger Features in Impala

- Resource-based and tag-based authorization policies
- Resource-based and tag-based column masking
- Row-level filtering is enabled by default

## Using Resource-based Authorization Policies for Impala

In the Ranger Service Manager, you can use the Hadoop SQL preloaded resource-based and tag-based services and policies to authorize access to Impala:



You can configure services for Impala as described in [Using Ranger to Provide Authorization in Cloudera](#).

For example, you can edit the all-global policy for an Impala user:



For information about using tag-based policies, see [Tag-based column masking in Hive with Ranger policies](#).

## Privilege Model

You set up privileges through the GRANT and REVOKE statements in either Impala or Hive. Then both components use those same privileges automatically.

By default, when authorization is not enabled, Impala does all read and write operations with the privileges of the `impala` user, which is suitable for a development/test environment but not for a secure production environment. When authorization is enabled, Impala uses the OS user ID of the user who runs `impala-shell` or other client programs, and associates various privileges with each user.

Privileges can be granted on different resources in the schema and are associated with a level in the resource hierarchy. A privilege on a particular resource automatically inherits the same privilege of its parent.

The resource hierarchy is:

```


Server
  URI
  Database
    Table
      Column
      Function

```

The table-level privileges apply to views as well. Anywhere you specify a table name, you can specify a view name instead.

You can specify privileges for individual columns.

The table below lists the minimum level of privileges and the scope required to run SQL statements in Impala. The following notations are used:

- The SERVER resource type in Ranger implies all databases, all tables, all columns, all UDFs, and all URLs.
-  **Note:** For Impala server level access, you must add the user to the below listed default policies in Ranger:
  - all - database, table, column
  - all - database, udf
  - all - url
- ANY denotes the CREATE, ALTER, DROP, SELECT, INSERT, or REFRESH privilege.
- ALL privilege denotes the SELECT, INSERT, CREATE, ALTER, DROP, and REFRESH privileges.
- VIEW\_METADATA privilege denotes the SELECT, INSERT, or REFRESH privileges.
- The parent levels of the specified scope are implicitly supported. For example, if a privilege is listed with the TABLE scope, the same privilege granted on DATABASE and SERVER will allow the user to run that specific SQL statement on TABLE.



**Note:** For an "impala" user to access any managed tables under the warehouse root directory or external tables with directories outside the warehouse root directory, the "impala" user must have HDFS RW access to the underlying paths that are referenced in a query.

Without this HDFS RW access, you may see Impala queries failing with HDFS errors such as this:

Error(13): Permission denied

Root cause: RemoteException: Permission denied: user=impala, access=WRITE, inode="/path/external/table":hive:supergroup:drwxr-xr-x

For example, to be able to run CREATE VIEW, you need the CREATE privilege on the database and the SELECT privilege on the source table.

SQL Statement	Privileges	Object Type / Resource Type
SELECT	SELECT	TABLE

WITH SELECT	SELECT	TABLE
EXPLAIN SELECT	SELECT	TABLE
INSERT	INSERT	TABLE
EXPLAIN INSERT	INSERT	TABLE
TRUNCATE	INSERT	TABLE
LOAD	INSERT	TABLE
	ALL	URI
CREATE DATABASE	CREATE	SERVER
CREATE DATABASE LOCATION	CREATE	SERVER
	ALL	URI
CREATE TABLE	CREATE	DATABASE
CREATE TABLE LIKE	CREATE	DATABASE
	VIEW_METADATA	TABLE
CREATE TABLE AS SELECT	CREATE	DATABASE
	INSERT	DATABASE
	SELECT	TABLE
EXPLAIN CREATE TABLE AS SELECT	CREATE	DATABASE
	INSERT	DATABASE
	SELECT	TABLE
CREATE TABLE LOCATION	CREATE	TABLE
	ALL	URI
CREATE VIEW	CREATE	DATABASE
	SELECT	TABLE
ALTER DATABASE SET OWNER	ALL WITH GRANT	DATABASE
ALTER TABLE	ALL	TABLE
ALTER TABLE SET LOCATION	ALL	TABLE
	ALL	URI
ALTER TABLE RENAME	CREATE	DATABASE
	ALL	TABLE
ALTER TABLE SET OWNER	ALL WITH GRANT	TABLE
ALTER VIEW	ALL	TABLE
	SELECT	TABLE
ALTER VIEW RENAME	CREATE	DATABASE
	ALL	TABLE
ALTER VIEW SET OWNER	ALL WITH GRANT	VIEW
DROP DATABASE	ALL	DATABASE
DROP TABLE	ALL	TABLE
DROP VIEW	ALL	TABLE
CREATE FUNCTION	CREATE	DATABASE
	ALL	URI

DROP FUNCTION	ALL	DATABASE
COMPUTE STATS	ALL	TABLE
DROP STATS	ALL	TABLE
INVALIDATE METADATA	REFRESH	SERVER
INVALIDATE METADATA <table>	REFRESH	TABLE
REFRESH <table>	REFRESH	TABLE
REFRESH AUTHORIZATION	REFRESH	SERVER
REFRESH FUNCTIONS	REFRESH	DATABASE
COMMENT ON DATABASE	ALL	DATABASE
COMMENT ON TABLE	ALL	TABLE
COMMENT ON VIEW	ALL	TABLE
COMMENT ON COLUMN	ALL	TABLE
DESCRIBE DATABASE	VIEW_METADATA	DATABASE
DESCRIBE <table/view>	VIEW_METADATA	TABLE
If the user has the SELECT privilege at the COLUMN level, only the columns the user has access will show.	SELECT	COLUMN
USE	ANY	TABLE
SHOW DATABASES	ANY	TABLE
SHOW TABLES	ANY	TABLE
SHOW FUNCTIONS	VIEW_METADATA	DATABASE
SHOW PARTITIONS	VIEW_METADATA	TABLE
SHOW TABLE STATS	VIEW_METADATA	TABLE
SHOW COLUMN STATS	VIEW_METADATA	TABLE
SHOW FILES	VIEW_METADATA	TABLE
SHOW CREATE TABLE	VIEW_METADATA	TABLE
SHOW CREATE VIEW	VIEW_METADATA	TABLE
SHOW CREATE FUNCTION	VIEW_METADATA	DATABASE
SHOW RANGE PARTITIONS (Kudu only)	VIEW_METADATA	TABLE
UPDATE (Kudu only)	ALL	TABLE
EXPLAIN UPDATE (Kudu only)	ALL	TABLE
UPSERT (Kudu only)	ALL	TABLE
WITH UPSERT (Kudu only)	ALL	TABLE
EXPLAIN UPSERT (Kudu only)	ALL	TABLE
DELETE	ALL	TABLE
EXPLAIN DELETE (Kudu only)	ALL	TABLE

The privileges not listed in the table above will be silently ignored by Impala.

### Changing Privileges in Impala

Privileges are managed via the GRANT and REVOKE SQL statements that require the Ranger service enabled.

Privileges can be also managed in Ranger UI. Especially, for attribute-based access control, Ranger UI is required to manage authorization.

Impala authorization policies are listed in the Hive service section in Ranger UI.

REFRESH AUTHORIZATION is not required when you make the changes to privileges within Impala. The changes are automatically propagated.

### Changing Privileges from Outside of Impala

If you make a change to privileges in Ranger from outside of Impala, e.g. adding a user, removing a user, modifying privileges, there are two options to propagate the change:

- Use the `ranger.plugin.hive.policy.pollIntervalMs` property to specify how often to do a Ranger refresh. The property is specified in `ranger-impala-security.xml` in the `conf` directory under your Impala home directory.
- Run the REFRESH AUTHORIZATION statement to force a refresh.



**Warning:** As INVALIDATE METADATA is an expensive operation, you should use it judiciously.

### Granting Privileges on URI

URIs represent the file paths you specify as part of statements such as CREATE EXTERNAL TABLE and LOAD DATA. Typically, you specify what look like UNIX paths, but these locations can also be prefixed with `hdfs://` to make clear that they are really URIs. To set privileges for a URI, specify the name of a directory, and the privilege applies to all the files in that directory.

URIs must start with `hdfs://`, `s3a://`, `adl://`, or `file://`. If a URI starts with an absolute path, the path will be appended to the default filesystem prefix. For example, if you specify:

```
GRANT ALL ON URI '/tmp' ;
```

The above statement effectively becomes the following where the default filesystem is HDFS.

```
GRANT ALL ON URI 'hdfs://localhost:20500/tmp' ;
```

When defining URIs for HDFS, you must also specify the NameNode. For example:

```
GRANT ALL ON URI file:///path/to/dir TO <principal>
GRANT ALL ON URI hdfs://namenode:port/path/to/dir TO <principal>
```



**Warning:** Because the NameNode host and port must be specified, it is strongly recommended that you use High Availability (HA). This ensures that the URI will remain constant even if the NameNode changes. For example:

```
GRANT ALL ON URI hdfs://ha-nn-uri/path/to/dir TO <principal>
```



**Note:**

```
<principal> can be a user, or a group
```



**Note:**

- If a user uses Impala SQL engine to access the resource of the specified URL/URI provided as part of the SQL statement and as an admin if you must deny access permissions to this location for this particular user, you must add the permission "All" in the "HADOOP SQL" policy.
- However, if the user is using Hive as the execution engine and to deny access permission to a location, then you must add both the "Read" and "Write" permissions in the field of 'Permissions' for the corresponding deny conditions.
- If the same Ranger policy is shared by both Hive and Impala, then you must add "All", "Read", and "Write" to the field of 'Permissions' to enforce the policy.

## Object Ownership

Object ownership for tables, views and databases is enabled by default in Impala.

To define owner specific privileges, go to Ranger UI and define appropriate policies on the {OWNER} user.

The CREATE statements implicitly make the user running the statement the owner of the object. For example, if *USER A* creates a database, *FOO*, via the CREATE DATABASE statement, *USER A* now owns the *FOO* database and is authorized to perform any operation on the *FOO* database.

An ownership can be transferred to another user or role via the ALTER DATABASE, ALTER TABLE, or ALTER VIEW with the SET OWNER clause.



**Note:** Currently, due to a known issue ([IMPALA-8937](#)), until the ownership information is fully loaded in the coordinator catalog cache, the owner of a table might not be able to see the table when executing the SHOW TABLES statement. The owner can still query the table.

## Ranger Column Masking for Impala

Ranger column masking hides sensitive columnar data in Impala query output. For example, you can define a policy that reveals only the first or last four characters of column data. Column masking is enabled by default. To disable column masking, modify the following configuration property in all coordinators:

```
--enable_column_masking=false
```

This flag might be removed in a future release. The Impala behavior mimics Hive behavior with respect to column masking.

The following table lists all Impala-supported, built-in mask types for defining column masking in a policy using the Ranger REST API or Ranger UI:

Type	Name	Description	Transformer
MASK	Redact	Replace lowercase with 'x', uppercase with 'X', digits with '0'	mask({col})
MASK_SHOW_LAST_4	Partial mask: show last 4	Show last 4 characters; replace rest with 'x'	mask_show_last_n({col}, 4, 'x', 'x', 'x', -1, '1')
MASK_SHOW_FIRST_4	Partial mask: show first 4	Show first 4 characters; replace rest with 'x'	mask_show_first_n({col}, 4, 'x', 'x', 'x', -1, '1')
MASK_HASH	Hash	Hash the value	mask_hash({col})
MASK_NULL	Nullify	Replace with NULL	N/A
MASK_NONE	Unmasked (retain original value)	No masking	N/A
MASK_DATE_SHOW_YEAR	Date: show only year	Date: show only year	mask({col}, 'x', 'x', 'x', -1, '1', 1, 0, -1)
CUSTOM	Custom	Custom	N/A

The table includes the mask name as it appears in the Ranger UI.

Select Masking Option

☐ Redact
 ☐ Partial mask: show last 4
 ☐ Partial mask: show first 4
 ☐ Hash
 ☐ Nullify
 ☐ Unmasked (retain original value)
 ☐ Date: show only year
 ☐ Custom

☒
☐

### Ranger Column Masking Limitations in Impala

- Column masking introduces unused columns during the query analysis and, consequently, additional SELECT privileges checks on all columns of the masked table.
- Impala might produce more than one audit log entry for a column involved in a column masking policy under all of these conditions: the column appears in multiple places in a query, the query is rewritten, or the query is re-analyzed.
- Column masking policies, shared between Hive and Impala, might be affected by SQL/UDF differences between Hive and Impala, as shown in the following example.

For instance, UTF-8 strings containing non-ASCII characters are not guaranteed to work properly. Suppose a column masking policy masks the last two characters of a string: `s => mask_last_n(s, 2, 'x', 'x', 'x', 'x')`. Applying this policy, Hive properly masks `SQL##` to `SQLxx`, but the Impala masking produces `SQL##xx` because each Chinese character is encoded in 3 bytes. Impala and Hive do not handle different lengths in the same way.

### Limitations on Mask Functions

The mask functions in Hive are implemented through GenericUDFs. Even though Impala users can call Hive UDFs, Impala does not yet support Hive GenericUDFs, so you cannot use Hive's mask functions in Impala. However, Impala has builtin mask functions that are implemented through overloads. In Impala, when using mask functions, not all parameter combinations are supported. These mask functions are introduced in Impala 3.4

The following list includes all the implemented overloads.

- Overloads used by Ranger default masking policies,
- Overloads with simple arguments,
- Overload with all arguments in int type for full functionality. Char argument needs to be converted to their ASCII value.

To list the available overloads, use the following query:

```
show functions in _impala_builtins like "mask*";
```



**Note:** An error message that states "No matching function with signature: mask..." implies that Impala does not contain the corresponding overload.

### Related Information

[GRANT statement](#)

[REVOKE statement](#)[Apache Ranger documentation](#)

## Row-level filtering in Impala with Ranger policies

Row-level filtering policies are similar to other Ranger access policies. Apache Ranger row-level filtering policy allows to set access policies for rows when reading from a table.

You can use Apache Ranger row-level filtering policies to set access policies for rows when reading from a table. You can set filters for specific users, groups, and conditions. This release adds a new feature flag `enable_row_filtering` which is set to be true by default. To enable row-filtering feature you must have set the column masking flag `enable_column_masking` to true since the row-level filtering depends on the column masking implementation. You can use this flag `enable_row_filtering` to disable this experimental feature as required.



**Note:** Note that row filtering policies apply prior to any column masking policies, because column masking policies apply only on result data of the target table/view.

The following limitations apply when using row-level filters:

- Row filtering policies on nested tables can't be applied when nested collection columns (e.g. array, map columns) are used directly in the FROM clause. Such queries are currently forbidden.

This is an example of the currently supported version of a query on the table `my_tbl` (`id` int, `int_array` array<int>) that has a row filter "`id = 0`".

```
select a.item from my_tbl t, t.int_array a
```

However an equivalent query below is not currently supported since it uses the `int_array` column non-relatively.

```
select item from my_tbl.int_array
```

Such queries are forbidden and you must rewrite them to the supported format until further notice.

- For information on the steps to set the row-level filtering using Apache Ranger, see the link provided under Related Information.



**Note:** Hive features that are not supported by Impala should not be used in the filter since Impala can't evaluate them.

## AES encryption and decryption support

This topic describes Impala's support for Advanced Encryption Standard (AES) crypto functions, including the various supported modes, key sizes, and function behavior.

Advanced Encryption Standard (AES) is a widely-used and secure way to protect sensitive data. It works by using a secret key to scramble your data (plaintext) into a form that's unreadable (ciphertext). Impala's AES implementation supports two key sizes, 128-bit and 256-bit, to meet different security requirements.

### Key functions and supported modes

Impala provides `aes_encrypt()` and `aes_decrypt()` functions as entry points for encryption and decryption operations, handling the processes based on user-provided keys, AES modes, and initialization vectors (IVs). The implementation includes key length and initialization vectors vector size validation to ensure data integrity and confidentiality.

These functions use a key, an encryption mode, and a special initialization vector (IV) to keep your data secure. Impala supports various modes to give you flexibility.

- Encryption Modes:
  - AES-GCM
  - AES-CTR
  - AES-CFB
- Decryption Modes:
  - AES-GCM
  - AES-ECB
  - AES-CTR
  - AES-CFB

You can refer to the following table to get information about the available modes:

Mode	Description
AES-GCM (Galois/Counter Mode)	This mode combines the AES block cipher with the Galois/Counter Mode (GCM) for authenticated encryption. It provides both confidentiality and integrity, making it suitable for secure communication and storage. Due to its strong security properties and efficiency, it is the default mode.
AES-ECB (Electronic Codebook)	This is a basic mode of operation where each block of plaintext is encrypted independently with the same key. It is included for compatibility with legacy systems and is only supported for decryption.
AES-CTR (Counter Mode)	This mode turns a block cipher into a stream cipher by encrypting a counter value to produce a key stream. This key stream is then XORed with the plaintext to create the ciphertext, allowing for parallel encryption and decryption.
AES-CFB (Cipher Feedback Mode)	This mode uses ciphertext feedback to create a key stream. It operates on a block-by-block basis, where the previous ciphertext block is encrypted and then XORed with the plaintext to produce the next ciphertext block.

### Compatibility and error handling

For compatibility, AES-CTR and AES-CFB are provided as fallbacks for environments where AES-GCM might not be supported (for example, OpenSSL versions prior to 1.0.1).

If a user-provided mode is a valid member of `AES_CIPHER_MODE` but is not supported by the internal OpenSSL library, the library's default mode is chosen.

If a user provides an unsupported or invalid mode, the system returns an error.

## Encryption and Decryption Examples

This topic provides examples for encrypting and decrypting data using various AES modes and demonstrates common error scenarios.

### Basic Encryption and Decryption

This is the most common use case. You encrypt a string and then decrypt it to get the original value back.

**Syntax:**

```
aes_encrypt(string, key, mode, iv) aes_decrypt(binary, key, mode, iv)
```

### Encrypting a String with AES\_128\_GCM:

Use this query to encrypt a string. The base64encode function converts the binary encrypted output into a text string that's easy to store.

```
SELECT base64encode(aes_encrypt('ABC',
  '1234567890123456', 'AES_128_GCM', '1234567890123456'));
```

In this example:

- 'ABC': The string to encrypt.
- '1234567890123456': The 128-bit encryption key.
- 'AES\_128\_GCM': The encryption mode.
- '1234567890123456': The initialization vector (IV).

The base64encode function is used to convert the binary encrypted output into a text string that can be easily stored or displayed.

### Decrypting the Encrypted String:

To reverse the process, use base64decode to convert the text back to binary, and then aes\_decrypt with the same key, mode, and IV.

```
SELECT aes_decrypt(base64decode('x+am+BIqtrEK9FpC/
zrvpOycjQ=='), '1234567890123456', 'AES_128_GCM', '1234567890123456');
```

This is the reverse process. base64decode converts the text back to binary, and aes\_decrypt uses the same key, mode, and IV to restore the original string, 'ABC'.

### Using Different Modes

The examples showcase different AES modes, each with its own characteristics.

AES\_128\_ECB: A basic mode of operation.



**Note:** The examples show it's only supported for decryption, as an error is thrown when used for encryption.

-- Decrypting with ECB mode

```
select aes_decrypt(base64decode('y6Ss
+zCYObpCbGfWfyNWTw=='), '1234567890123456', 'AES_128_ECB', '');
-- RESULTS
'ABC'
```

AES\_256\_GCM: This mode requires a longer 256-bit key.



**Note:** The key string '12345678901234567890123456789012' is longer, matching the 256-bit requirement.

### Encrypt with AES\_256\_GCM

```
select base64encode(aes_encrypt('ABC',
  '12345678901234567890123456789012', 'AES_256_GCM', '1234567890123456'));
```

### Decrypt with AES\_256\_GCM

```
select aes_decrypt(base64decode('F/
DLkSwEikFOlqzXVCysylJX7Q=='), '12345678901234567890123456789012', 'AES_256_GCM', '1234567890123456789012');
```

### Common Error Scenarios

The examples also highlight important constraints and errors.

**NULL Values:** The examples show that if the key is NULL, an error will occur.

```
-- This will fail because the key is NULL
select
  base64encode(aes_encrypt('ABC', NULL, 'AES_256_GCM', '1234567890123456'));
```

**Invalid Mode:** Using an incorrect mode string will result in an error.

```
-- This will fail due to an invalid mode name
select base64encode(aes_encrypt('ABC',
  '12345678901234567890123456789012', 'AES_256_CTB', '1234567890123456'));
```

**Incorrect Key or IV Length:** The key must be either 128 or 256 bits long. The IV must also adhere to specific length requirements.

```
-- This will fail due to an incorrect key length
select base64encode(aes_encrypt('ABC',
  '123456789012345678901234567890121', 'AES_256_GCM', '1234567890123456'));
```

**Case Insensitivity:** The mode string is not case-sensitive.

```
-- This works, even with lowercase mode
select base64encode(aes_encrypt('ABC',
  '12345678901234567890123456789012', 'aes_256_gcm', '1234567890123456'));
```