

# Data Lake Compute

## SQL Syntax



## Copyright Notice

©2013–2026 Tencent Cloud. All rights reserved.

The complete copyright of this document, including all text, data, images, and other content, is solely and exclusively owned by Tencent Cloud Computing (Beijing) Co., Ltd. ("Tencent Cloud"); Without prior explicit written permission from Tencent Cloud, no entity shall reproduce, modify, use, plagiarize, or disseminate the entire or partial content of this document in any form. Such actions constitute an infringement of Tencent Cloud's copyright, and Tencent Cloud will take legal measures to pursue liability under the applicable laws.

## Trademark Notice



This trademark and its related service trademarks are owned by Tencent Cloud Computing (Beijing) Co., Ltd. and its affiliated companies ("Tencent Cloud"). The trademarks of third parties mentioned in this document are the property of their respective owners under the applicable laws. Without the written permission of Tencent Cloud and the relevant trademark rights owners, no entity shall use, reproduce, modify, disseminate, or copy the trademarks as mentioned above in any way. Any such actions will constitute an infringement of Tencent Cloud's and the relevant owners' trademark rights, and Tencent Cloud will take legal measures to pursue liability under the applicable laws.

## Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

## Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

# Contents

## SQL Syntax

### SQL Syntax Overview

### SuperSQL Syntax

#### SuperSQL Syntax Overview

### Uniform Syntax

#### Common Data Types

#### DDL Syntax

CREATE DATABASE

SHOW DATABASES

DESCRIBE DATABASE

ALTER DATABASE

ALTER DATABASE SET DBPROPERTIES

ALTER DATABASE SET LOCATION

DROP DATABASE

CREATE TABLE

REPLACE TABLE AS SELECT

SHOW TABLES

SHOW CREATE TABLE

SHOW TBLPROPERTIES

DESCRIBE TABLE

SHOW COLUMNS IN TABLE

ALTER TABLE

ALTER TABLE ADD COLUMNS

ALTER TABLE ADD COLUMN AFTER/FIRST

ALTER TABLE ... RENAME COLUMN

ALTER TABLE DROP COLUMN

ALTER TABLE ADD PARTITION

SHOW PARTITIONS

ALTER TABLE DROP PARTITION

ALTER TABLE ADD PARTITION FIELD

ALTER TABLE DROP PARTITION FIELD

ALTER TABLE SET TBLPROPERTIES

ALTER TABLE SET LOCATION

ALTER TABLE ... WRITE ORDERED BY

ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION

- ALTER TABLE ... SET IDENTIFIER FIELDS
  - ALTER TABLE ... DROP IDENTIFIER FIELDS
- MSCK REPAIR TABLE
- ANALYZE TABLES
- DROP TABLE
- EXPLAIN
- CALL STATEMENT
- CREATE VIEW AS
- SHOW VIEWS
- DESCRIBE VIEW
- SHOW CREATE VIEW
- SHOW COLUMNS IN VIEW
- ALTER VIEW
  - ALTER VIEW RENAME TO
  - ALTER VIEW SET TBLPROPERTIES
- DROP VIEW
- CREATE FUNCTION
- SHOW FUNCTION
- DROP FUNCTION
- DML syntax
  - INSERT STATEMENT
  - INSERT INTO
  - INSERT OVERWRITE
  - MERGE INTO
  - UPDATE
  - DELETE STATEMENT
- TABLE METADATA
- DQL syntax
  - SELECT STATEMENT
- Iceberg Table Syntax
  - DDL Syntax
  - DML Syntax
  - DQL Syntax
  - Procedure
  - Syntax Differences Between Iceberg External Tables and Native Tables
- SQL Implicit Conversion
- Function
  - Unified Function



**Unified Function Overview**

**Binary Functions**

**Bitwise Operation Functions**

**Aggregate Functions**

**Date and Time Functions**

**JSON Functions**

**Math Function**

**String Functions**

**Aggregate Functions**

**Window Functions**

**Other Functions**

**Presto Built-in Functions**

**Hive Function Reference**

**Overview of Standard Spark Syntax**

**Overview of Standard Presto Syntax**

**Reserved Words**

# SQL Syntax

## SQL Syntax Overview

Last updated: 2026-05-20 17:53:04

[SuperSQL Syntax Overview](#)

[Standard Spark Syntax Overview](#)

[Standard Presto Syntax Overview](#)

# SuperSQL Syntax

## SuperSQL Syntax Overview

Last updated: 2026-05-20 17:53:39

DLC can run almost seamlessly on both DLC Serverless Spark and DLC Serverless Presto engines using a standard set of SQL syntax. Its metadata, analytical syntax, and functions are largely compatible with Hive and Spark syntax, and custom functions are supported. For the scope of system built-in functions, refer to [Unified Functions](#). To use Presto built-in functions, see [Presto Built-in Functions](#) for usage and supported scope. If you need to query and analyze data from external Iceberg tables in DLC, note that some syntax differs from that for native tables. For details, see [Syntax Differences Between Iceberg External Tables and Native Tables](#).

}

### DDL Syntax

#### Database-Related Syntax

| Purpose                                           | Syntax                                          |
|---------------------------------------------------|-------------------------------------------------|
| Creating a database                               | <a href="#">CREATE DATABASE</a>                 |
| Displaying all databases defined in the metadata. | <a href="#">SHOW DATABASES</a>                  |
| Viewing database properties                       | <a href="#">DESCRIBE DATABASE</a>               |
| Modifying database properties                     | <a href="#">ALTER DATABASE SET DBPROPERTIES</a> |
| Changing database storage location                | <a href="#">ALTER DATABASE SET LOCATION</a>     |
| Delete a database.                                | <a href="#">DROP DATABASE</a>                   |

#### Data Table-Related Syntax

| Purpose                   | Syntax                                   |
|---------------------------|------------------------------------------|
| Creating a data table     | <a href="#">CREATE TABLE</a>             |
| Updating a table snapshot | <a href="#">REPLACE TABLE AS SELESCT</a> |

|                                                           |                                                                |
|-----------------------------------------------------------|----------------------------------------------------------------|
| Querying data table creation information                  | <a href="#">SHOW CREATE TABLE</a>                              |
| Querying table properties                                 | <a href="#">SHOW TBLPROPERTIES</a>                             |
| Querying all tables in a database                         | <a href="#">SHOW TABLES</a>                                    |
| Viewing data table column information and metadata        | <a href="#">DESCRIBE TABLE</a>                                 |
| Querying data table column information                    | <a href="#">SHOW COLUMNS IN TABLE</a>                          |
| Adding columns to a data table                            | <a href="#">ALTER TABLE ADD COLUMNS</a>                        |
| Adding a new column to a data table                       | <a href="#">ALTER TABLE ADD COLUMN AFTER/FIRST</a>             |
| Renaming a field                                          | <a href="#">ALTER TABLE ... RENAME COLUMN</a>                  |
| Dropping a column from a data table                       | <a href="#">ALTER TABLE DROP COLUMN</a>                        |
| Adding partition information to a data table              | <a href="#">ALTER TABLE ADD PARTATION</a>                      |
| Listing table partitions                                  | <a href="#">SHOW PARTITIONS</a>                                |
| Dropping partition information from a data table          | <a href="#">ALTER TABLE DROP PARTITION</a>                     |
| Adding a partition field to an Iceberg table              | <a href="#">ALTER TABLE ADD PARTITION FIELD</a>                |
| Dropping a partition field from an Iceberg table          | <a href="#">ALTER TABLE DROP PARTITION FIELD</a>               |
| Modifying data table properties                           | <a href="#">ALTER TABLE SET TBLPROPERTIES</a>                  |
| Changing data table storage location                      | <a href="#">ALTER TABLE SET LOCATION</a>                       |
| Modifying the sort order of table data                    | <a href="#">ALTER TABLE ... WRITE ORDERED BY</a>               |
| Modifying the distribution policy for a partitioned table | <a href="#">ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION</a> |

|                                                                        |                                                        |
|------------------------------------------------------------------------|--------------------------------------------------------|
| Adding identifier fields property                                      | <a href="#">ALTER TABLE ... SET IDENTIFIER FIELDS</a>  |
| Dropping identifier fields property                                    | <a href="#">ALTER TABLE ... DROP IDENTIFIER FIELDS</a> |
| Updating partition information                                         | <a href="#">MSCK REPAIR TABLE</a>                      |
| Analyzing a data table                                                 | <a href="#">ANALYZE TABLES</a>                         |
| Dropping a metadata table                                              | <a href="#">DROP TABLE</a>                             |
| Displaying the logical or physical plan for executing an sql statement | <a href="#">EXPLAIN</a>                                |
| Calling a table stored procedure                                       | <a href="#">CALL STATEMENT</a>                         |

## View-Related Syntax

| Purpose                                | Syntax                                       |
|----------------------------------------|----------------------------------------------|
| Creating a view from a select result   | <a href="#">CREATE VIEW AS</a>               |
| Querying views in a database           | <a href="#">SHOW VIEWS</a>                   |
| Viewing view column information        | <a href="#">DESCRIBE VIEW</a>                |
| Displaying the view creation statement | <a href="#">SHOW CREATE VIEW</a>             |
| Viewing view column information        | <a href="#">SHOW COLUMNS IN VIEW</a>         |
| Renaming a view                        | <a href="#">ALTER VIEW RENAME TO</a>         |
| Modifying view properties              | <a href="#">ALTER VIEW SET TBLPROPERTIES</a> |
| Deleting a View                        | <a href="#">DROP VIEW</a>                    |

## Function-Related Syntax

| Purpose             |                                 |
|---------------------|---------------------------------|
| Creating a Function | <a href="#">CREATE FUNCTION</a> |

|                                            |                               |
|--------------------------------------------|-------------------------------|
| Viewing the Syntax for Creating a Function | <a href="#">SHOW FUNCTION</a> |
| Deleting a Function                        | <a href="#">DROP FUNCTION</a> |

## DML Syntax

| Purpose                                                                                       |                                  |
|-----------------------------------------------------------------------------------------------|----------------------------------|
| Inserting a row of data                                                                       | <a href="#">INSERT STATEMENT</a> |
| Replacing a row of data                                                                       | <a href="#">INSERT OVERWRITE</a> |
| Row-level data update operation, which can be used to replace the INSERT OVERWRITE operation. | <a href="#">MERGE INTO</a>       |
| Querying Iceberg table metadata                                                               | <a href="#">TABLE METADATA</a>   |
| Inserting query results into a data table.                                                    | <a href="#">INSERT INTO</a>      |
| Deleting data from an Iceberg table.                                                          | <a href="#">DELETE STATEMENT</a> |
| Updating specified rows                                                                       | <a href="#">UPDATE</a>           |

## DQL Syntax

| Purpose    |                                  |
|------------|----------------------------------|
| Data query | <a href="#">SELECT STATEMENT</a> |

For related query reserved words, see [Reserved Words](#).

# Uniform Syntax

## Common Data Types

Last updated: 2026-05-20 17:54:11

| Data Type Category | Data Type                         | Description                                                                                               |
|--------------------|-----------------------------------|-----------------------------------------------------------------------------------------------------------|
| Numeric            | ByteType:BYTE, TINYINT            | A 1-byte signed integer with a value range from -128 to 127.                                              |
|                    | ShortType:SHORT, SMALLINT         | A 2-byte signed integer with a value range from -32768 to 3276.                                           |
|                    | IntegerType:INT, INTEGER          | A 4-byte signed integer with a value range from -2147483648 to 214748364.                                 |
|                    | LongType:LONG, BIGINT             | An 8-byte signed integer with a value range from -9223372036854775808 to 922337203685477580.              |
|                    | FloatType:FLOAT, REAL             | A 4-byte single-precision floating-point number.                                                          |
|                    | DoubleType:DOUBLE                 | An 8-byte double-precision floating-point number.                                                         |
|                    | DecimalType:DECIMAL, DEC, NUMERIC | A signed decimal number of arbitrary precision.                                                           |
| String             | StringType:STRING                 | String values, for example: 'abc'.                                                                        |
| Bytes              | BinaryType:BINARY                 | Byte sequence values.                                                                                     |
| Boolean            | BooleanType:BOOLEAN               | Boolean values, for example: true/false.                                                                  |
| Time               | DateType:DATE                     | A value composed of year, month, and day field values, without a time zone.<br>For example: '2023-10-01'. |

|               |                                                                            |                                                                                                                                                                                               |
|---------------|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | TimestampType:TIMESTAMP                                                    | <p>A value composed of year, month, day, hour, minute, and second field values, along with the local time zone.</p> <p>For example: '2023-10-01 23:59:59'.</p>                                |
| Complex types | ArrayType:ARRAY<element_type>                                              | <p>A sequence value composed of elements of the element_type type.</p> <p>For example: array&lt;int&gt;.</p>                                                                                  |
|               | MapType:MAP<key_type, value_type>                                          | <p>A value composed of a set of key-value pairs, where key_type specifies the type of the key and value_type specifies the type of the value.</p> <p>For example: map&lt;string, int&gt;.</p> |
|               | StructType:STRUCT<field1_name: field1_type, field2_name: field2_type, ...> | <p>A structured value composed of fields.</p> <p>For example: struct&lt;id:int, grade:string&gt;.</p>                                                                                         |



# DDL Syntax

## CREATE DATABASE

Last updated: 2026-05-20 18:00:15

### Description

- Supported engines: Presto, SparkSQL.
- Purpose: Creates a database. You can also specify or change the storage location for tables or partitioned tables.

### Syntax

```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] database_name
    [COMMENT 'database_comment']
    [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

### Parameter

- `DATABASE|SCHEMA` : They have the same meaning and can be used interchangeably.
- `database_name` : The database name.
- `database_comment` : The database comment.
- `[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]` : Configures database parameters in `key-value` pairs.

### Examples

```
-- Creates the database db only if a database with the same name does
not exist.
CREATE DATABASE IF NOT EXISTS db;

-- Creates the database db only if a database with the same name does
not exist, and includes comments and database properties.
CREATE DATABASE db
COMMENT 'db1_name'
WITH DBPROPERTIES('k1' = 'v1', 'k2' = 'v2');
```

# SHOW DATABASES

Last updated: 2026-05-20 18:00:49

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Lists all databases defined in this metadata. You can use DATABASES or SCHEMAS to perform the same query.

## Syntax

```
SHOW {DATABASES | SCHEMAS} [IN catalog_name] [LIKE 'regular_expression']
```

## Parameter

- `[IN catalog_name]` : The data source name.
- `[LIKE 'regular_expression']` : Filters out database names that match the pattern.

## Examples

```
SHOW DATABASES;  
SHOW DATABASES LIKE '.*analytics';  
SHOW DATABASES IN catalog1 LIKE '.*analytics';
```

# DESCRIBE DATABASE

Last updated: 2026-05-20 18:01:05

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: View database properties.

## Standard Syntax

```
DESCRIBE SCHEMA|DATABASE [EXTENDED] DB_NAME
```

## Parameter

- `SCHEMA|DATABASE` : Specify the library as SCHEMA or DATABASE.
- `EXTENDED` : Whether the library is EXTENDED.

## Examples

```
DESCRIBE DATABASE db_name;  
DESCRIBE DATABASE EXTENDED db_name;
```

# ALTER DATABASE ALTER DATABASE SET DBPROPERTIES

Last updated: 2026-05-20 18:01:37

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Adds one or more properties to a database. If a property already exists, it is overwritten.

## Standard Syntax

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES  
(property_name=property_value, ...)
```

## Parameter

`database_name` : The database name.

## Examples

```
-- Modify the database to set the "author" property.  
ALTER DATABASE product SET DBPROPERTIES ('author' = 'allen');
```

# ALTER DATABASE SET LOCATION

Last updated: 2026-05-20 18:02:04

## Description

- Supported engines: Presto.
- Purpose: Modifies the storage path of a database.

## Standard Syntax

```
ALTER (DATABASE | SCHEMA) database_name SET LOCATION hdfs_path
```

## Parameter

[database\_name] : The database name.

## Examples

```
ALTER DATABASE db01 SET LOCATION 'cosn:///new/path'
```

# DROP DATABASE

Last updated: 2026-05-20 18:02:26

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Deletes a specified database.

## Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] database_name [RESTRICT | CASCADE]
```

## Parameter

- `DATABASE | SCHEMA` : They have the same meaning and can be used interchangeably.
- `database_name` : The database name.
- `RESTRICT` : If the database contains tables, it will not be deleted. If not filled in, this mode is the default.
- `CASCADE` : Forces the deletion of all database tables.

## Examples

```
-- Deletes the database and the tables it contains.  
DROP DATABASE test CASCADE;  
  
-- Deletes a database using the IF EXISTS statement.  
DROP DATABASE IF EXISTS test;
```

# CREATE TABLE

Last updated: 2026-05-20 18:02:54

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Creates a table with specified attributes and supports the CREATE TABLE AS syntax.
- Table storage path: The storage path for a table can be specified to a COS directory but not to a file.

## External Table Syntax

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
USING data_source
    [ COMMENT table_comment ]
    [ OPTIONS ( 'key1'='value1', 'key2'='value2' ) ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
```

## Parameters

**USING data\_source** : Specifies the input data format when a table is created. Currently supported formats include CSV, ORC, PARQUET, and ICEBERG.

**table\_identifier** : Specifies the table name and supports a three-part format, for example, catalog.database.table.

**COMMENT** : The description of the table.

**OPTIONS** : Additional parameters supported by USING data\_source, used for parameter injection during storage.

**PARTITIONED BY** : Creates partitions based on specified columns.

**LOCATION path** : The storage path for the data table.

**TBLPROPERTIES** : A set of key-value pairs used to specify table parameters.

## Detailed Description of USING and OPTIONS Parameters

## USING CSV

The following configurations are supported for CSV data tables.

| Supported Keys for OPTIONS | Default value | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sep or delimiter           | ,             | The delimiter between columns in csv storage, defaulting to a comma                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| mode                       | PERMISSIVE    | Defining the handling mode when data conversion does not meet expectations.<br>PERMISSIVE: A more lenient mode, which is the default. It attempts to convert each row of data. For example, if a row has extra columns, it automatically takes only the required columns.<br>DROPMALFORMED: Discards data that does not meet expectations. For example, if a row has extra columns, that row is discarded.<br>FAILFAST: Strictly requires csv format. If a row does not meet expectations, it fails. For example, if there are extra columns. |
| encoding or charset        | UTF-8         | String encoding format.<br>For example: UTF-8, US-ASCII, ISO-8859-1, UTF-16BE, UTF-16LE, UTF-16                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| quote                      | \"            | Whether the quote is a single quote or a double quote. Pay attention to using escape characters.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| escape                     | \\            | Escape character. Pay attention to using escape characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| charToEscapeQuoteEscaping  | -             | Characters that need to be escaped inside quotes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| comment                    | \u0000        | Remarks                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| header                     | false         | Whether a header exists                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| inferSchema                | false         | Infer the data type of each column. If not inferred, each column is treated as a                                                                                                                                                                                                                                                                                                                                                                                                                                                              |



|                           |                              |                                                                                                                                                                       |
|---------------------------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           |                              | string.                                                                                                                                                               |
| ignoreLeadingWhiteSpace   | read: false<br>write: true   | Ignoring leading whitespace                                                                                                                                           |
| ignoreTrailingWhiteSpace  | read: false<br>write: true   | Ignoring trailing whitespace                                                                                                                                          |
| columnNameOfCorruptRecord | _corrupt_record              | The column name for records that cannot be parsed. This parameter is affected by spark.sql.columnNameOfCorruptRecord, with table configuration taking precedence.     |
| nullValue                 | –                            | The storage format for null values. It defaults to an empty string, in which case it is written according to the emptyValue configuration.                            |
| nanValue                  | NaN                          | The storage format for non-numeric values                                                                                                                             |
| positiveInf               | Inf                          | The storage format for positive infinity                                                                                                                              |
| negativeInf               | –Inf                         | The storage format for negative infinity                                                                                                                              |
| compression or codec      | –                            | The class name of the compression algorithm. By default, no compression is applied. Abbreviations can be used, such as bzip2, deflate, gzip, lz4, and snappy.         |
| timeZone                  | system default time zone     | The default time zone. The value of this parameter is affected by spark.sql.session.timeZone, for example, Asia/Shanghai, with table configuration taking precedence. |
| locale                    | en-US                        | Language type                                                                                                                                                         |
| dateFormat                | yyyy-MM-dd                   | The default date format                                                                                                                                               |
| timestampFormat           | yyyy-MM-dd'T'HH:mm:ss.SSSXXX | The default timestamp format. In non-LEGACY mode, it is yyyy-MM-dd'T'HH:mm:ss[.SSS][XXX].                                                                             |
| multiLine                 | false                        | Allowing multiple lines                                                                                                                                               |

|                         |                     |                                                                                                                                                                                                                                                                                               |
|-------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| maxColumns              | 20480               | Maximum number of columns                                                                                                                                                                                                                                                                     |
| maxCharsPerColumn       | -1                  | Maximum characters per column. A value of -1 indicates no limit.                                                                                                                                                                                                                              |
| escapeQuotes            | true                | Escaping quotes                                                                                                                                                                                                                                                                               |
| quoteAll                | quoteAll            | Adding quotes to all content during writing                                                                                                                                                                                                                                                   |
| samplingRatio           | 1.0                 | Sampling ratio                                                                                                                                                                                                                                                                                |
| enforceSchema           | true                | Forces reading using the specified schema, ignoring the header definition.                                                                                                                                                                                                                    |
| emptyValue              | read:<br>write: \"\ | The read/write format for null values                                                                                                                                                                                                                                                         |
| lineSep                 | -                   | Line break                                                                                                                                                                                                                                                                                    |
| inputBufferSize         | -                   | The buffer size for reading. This parameter can be affected by spark.sql.csv.parser.inputBufferSize, with table configuration taking precedence.                                                                                                                                              |
| unescapedQuote Handling | STOP_AT_DELIMITER   | The handling policy when unescaped quotes are detected.<br>STOP_AT_DELIMITER: Stops reading at the delimiter<br>BACK_TO_DELIMITER: Rolls back to the delimiter<br>STOP_AT_CLOSING_QUOTE: Stops reading at the next quote<br>SKIP_VALUE: Skips the column data<br>RAISE_ERROR: Raises an error |

USING ORC

The following configurations are supported for ORC data tables:

| Supported Keys for OPTIONS | Default value | Meaning |
|----------------------------|---------------|---------|
|----------------------------|---------------|---------|

|                                |        |                                                                                                                                                                                               |
|--------------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| compression or<br>orc.compress | snappy | The compression algorithm. It supports the abbreviations snappy/zlib/lzo/lz3/zstd. This parameter is affected by spark.sql.orc.compression.codec, with the table parameter taking precedence. |
| mergeSchema                    | false  | Merging schemas. This parameter is affected by spark.sql.orc.mergeSchema, with the table parameter taking precedence.                                                                         |

When HiveRead and HiveWriter are used (with spark.sql.hive.convertMetastoreOrc=false) for reading and writing, native Orc configurations are also supported by OPTIONS. For details, see [LanguageManual ORC](#).

## USING PARQUET

Most parameters related to PARQUET data tables can be configured through Spark conf, and it is also recommended to configure them from Spark conf. The following configurations are also supported by options:

| Supported Keys for<br>OPTIONS         | Default value | Meaning                                                                                                                                                           |
|---------------------------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| compression or<br>parquet.compression | snappy        | The compression algorithm. It uses snappy by default, is affected by the parameter spark.sql.parquet.compression.codec, and the table parameter takes precedence. |
| mergeSchema                           | false         | Whether to merge schemas. It is affected by the parameter                                                                                                         |

|                           |                  |                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           |                  | <p>spark.sql.parquet.mergeSchema, with the table parameter taking precedence.</p>                                                                                                                                                                                                                                                                                                                                 |
| <p>datetimeRebaseMode</p> | <p>EXCEPTION</p> | <p>The date conversion policy when parquet files are written. The LEGACY mode converts dates to the Gregorian calendar, the CORRECTED mode does not convert dates to the Gregorian calendar, and the EXCEPTION mode reports an error when dates in different formats are encountered. It is affected by the parameter spark.sql.parquet.datetimeRebaseModeInRead, with the table parameter taking precedence.</p> |
| <p>int96RebaseMode</p>    | <p>EXCEPTION</p> | <p>The time conversion policy when parquet files are read. The LEGACY mode converts time to the Gregorian calendar, the CORRECTED mode does not convert time, and the EXCEPTION mode reports an error when time in different formats is encountered. It is affected by the parameter spark.sql.parquet.int96RebaseModeInRead, with the table parameter taking precedence.</p>                                     |

When HiveRead and HiveWriter are used (with spark.sql.hive.convertMetastoreParquet=false) for reading and writing, native Parquet

configurations can also be specified in OPTIONS. For reference, see [Hadoop integration](#).

## Example

```
CREATE TABLE dempts(  
  id bigint COMMENT 'id number',  
  num int,  
  eno float,  
  dno double,  
  cno decimal(9,3),  
  flag boolean,  
  data string,  
  ts_year timestamp,  
  date_month date,  
  bno binary,  
  point struct<x: double, y: double>,  
  points array<struct<x: double, y: double>>,  
  pointmaps map<struct<x: int>, struct<a: int>>  
)  
  
USING iceberg  
COMMENT 'table documentation'  
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month),  
identity(bno), bucket(3,num), truncate(10,data))  
LOCATION '/warehouse/db_001/dempts'  
TBLPROPERTIES ('write.format.default'='orc');
```

## FAQs

In the CREATE\_TABLE syntax, the keywords USING in Spark and STORED AS in Hive differ, which may cause unexpected file formats or reading behavior after table creation. A special note is provided here:

- **USING DATA\_SOURCE:** A Spark syntax keyword that specifies the data source format used as input when a table is created. This directly determines the file format and reading method under the table's Location. Acceptable values include CSV, TXT, Iceberg, Parquet, Orc, and others.
- **STORED AS FILE\_FORMAT:** A Hive syntax keyword used to create a table in HIVE format, indicating the format of the data files stored in the table. Acceptable values include TXT, Parquet, Orc, and others. This syntax is not recommended, as it may cause Spark's native reader/writer to fail to support it, for example, CSV is not supported.

# Native Table Iceberg Syntax

## Note:

This syntax only supports creating native tables.

## Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

## Parameters

`table_identifier` : Supports a three-part format, such as `catalog.db.name`.

## Schemas and Data Types

```
col_type
: primitive_type
| nested_type

primitive_type
: boolean
| int/integer
| long/bigint
| float
| double
| decimal(p,s), where p = maximum number of digits, s = maximum number
of decimal places, and s <= p <= 38.
| date
| timestamp and timestamp with timezone are supported, while time and
without timezone are not supported.
| string, which can also correspond to the Iceberg uuid type.
| binary, which can also correspond to the Iceberg fixed type.

nested_type
: struct
| list
```

```
| map
```

## Partition Transforms

```
transform
  : identity, which supports any type. DLC does not support this
  conversion.
  | bucket[N], which uses hash mod N for bucketing, supports the
  following column types: int, long, decimal, date, timestamp, string,
  binary.
  | truncate[L], which performs bucketing by truncating L, supports the
  following column types: int, long, decimal, string.
  | years, which represents the year, supports the following column
  types: date, timestamp.
  | months, which represents the month, supports the following column
  types: date, timestamp.
  | days/date, which represents the day, supports the following column
  types: date, timestamp.
  | hours/date_hour, which represents the hour, supports the following
  column type: timestamp.
```

## Example

```
CREATE TABLE dempts(
  id bigint COMMENT 'id number',
  num int,
  eno float,
  dno double,
  cno decimal(9,3),
  flag boolean,
  data string,
  ts_year timestamp,
  date_month date,
  bno binary,
  point struct<x: double, y: double>,
  points array<struct<x: double, y: double>>,
  pointmaps map<struct<x: int>, struct<a: int>>
)
COMMENT 'table documentation'
```

```
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month),  
identity(bno), bucket(3,num), truncate(10,data));
```



# REPLACE TABLE AS SELECT

Last updated: 2026-05-20 18:03:23

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Updates or replaces a table snapshot from the target table while preserving the table history.

## Syntax

```
CREATE [OR REPLACE] TABLE table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

## Examples

```
CREATE OR REPLACE TABLE dempts_replace
USING iceberg
COMMENT 'table create as replace'
PARTITIONED BY (eno, dno)
TBLPROPERTIES ('write.format.default'='avro')
LOCATION '/warehouse/db_001/dempts_replace'
AS SELECT * from dempts;
```

# SHOW TABLES

Last updated: 2026-05-20 18:03:46

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Lists all base tables and views in the database.

## Syntax

```
SHOW TABLES [IN database_name] ['regular_expression']
```

## Parameter

- `[IN database_name]` : Specifies the name of the database from which to list tables. If the `database_name` is omitted, the database in the current context is assumed.
- `['regular_expression']` : Filters the table list to show only the tables that match the specified regular expression. You can only use the wildcard `*` to represent any character, or to indicate alternatives between characters.

## Examples

```
SHOW TABLES IN sampled;db;
```

```
SHOW TABLES IN sampled;db '*flights*';
```

# SHOW CREATE TABLE

Last updated: 2026-05-20 18:04:14

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Analyzes the existing table `table\_name` and queries its creation information.

## Syntax

```
SHOW CREATE TABLE [catalog_name.] [db_name.] table_name
```

## Parameter

TABLE [db\_name.] table\_name

- db\_name : The database name.
- table\_name : The table name.

## Examples

```
SHOW CREATE TABLE tbl;
```

# SHOW TBLPROPERTIES

Last updated: 2026-05-20 18:04:44

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Lists the table properties of named tables.

## Syntax

```
SHOW TBLPROPERTIES table_name [('property_name')]
```

## Parameter

`[('property_name')]` : If it is included, only the property name and value are listed.

## Examples

```
SHOW TBLPROPERTIES tb1;
```

```
SHOW TBLPROPERTIES orders('tb1');
```

# DESCRIBE TABLE

Last updated: 2026-05-20 18:05:08

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Views the column information and metadata of a data table.

## Standard Syntax

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION  
partition_spec];
```

## Parameter

- [EXTENDED | FORMATTED] : Specifies the formatted form of the table.
- table\_name : The required table name.
- [PARTITION partition\_spec] : The partition list of the table.

## Examples

```
DESCRIBE tbl;  
DESCRIBE FORMATTED tbl PARTITION (date_id = '2019-01-07');
```

# SHOW COLUMNS IN TABLE

Last updated: 2026-05-20 18:12:02

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Views the column information of a data table.

## Syntax

```
SHOW COLUMNS IN table_name
```

## Parameter

`table_name` : The name of the data table.

## Examples

```
SHOW COLUMNS IN clicks;
```

# ALTER TABLE

## ALTER TABLE ADD COLUMNS

Last updated: 2026-05-20 18:12:36

### Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Modifies data table properties.

### Standard Syntax

```
ALTER TABLE table_name
[PARTITION
  (partition_col1_name = partition_col1_value
  [,partition_col2_name = partition_col2_value][,...])]
ADD COLUMNS (col_name data_type) [RESTRICT | CASCADE]
```

### Parameter

- `table_name` : The required table name.
- `partition_col1_name` : The partition name.
- `partition_col1_value` : The partition value.
- `col_name` : The column name to be added.
- `data_type` : The data type of the column to be added.

### Examples

```
ALTER TABLE events ADD COLUMNS (eventowner string);

ALTER TABLE events ADD COLUMNS (eventowner string) CASCADE;

// The ALTER TABLE PARTITION ADD COLUMNS syntax is supported only for
DLC native tables.
ALTER TABLE events PARTITION (year='2021') ADD COLUMNS (event string);
```

**Note:**

If a table is created using the `ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'` format, columns cannot be added after table creation. When a table is created with `JsonSerDe`, verify the table structure as thoroughly as possible. If columns must be added, consider dropping and recreating the table.



# ALTER TABLE ADD COLUMN AFTER/FIRST

Last updated: 2026-05-20 18:12:53

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Adds columns to a data table.

## Standard Syntax

```
ALTER TABLE table_name ADD COLUMN column_name1 column_type [COMMENT  
col_comment] [FIRST|AFTER column_name2]
```

## Parameter

- `table_name` : The name of the table to be modified.
- `column_name1` : The column to be added.
- `column_type` : The data type of the column to be added.
- `col_comment` : The comment to be added to the column.
- `column_name2` : The column to be added is placed after this column.

## Examples

```
ALTER TABLE `TBL` ADD COLUMN `COL2` STRING COMMENT 'test' AFTER `COL1`
```

```
ALTER TABLE `TBL` ADD COLUMN `COL2` STRING COMMENT 'test' FIRST
```

# ALTER TABLE ... RENAME COLUMN

Last updated: 2026-05-20 18:13:14

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Changes the field name.

## Syntax

```
ALTER TABLE table_identifier  
RENAME COLUMN old_column_name TO new_column_name
```

## Parameter

- `table_identifier` : The name of the data table.
- `old_column_name` : The name of the field to be changed.
- `new_column_name` : The new name of the field after the change.

## Examples

```
alter table iceberg_rename rename column id to id_2
```

# ALTER TABLE DROP COLUMN

Last updated: 2026-05-20 18:13:36

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables.
- Purpose: Deletes a specified field from a data table.

## Standard Syntax

```
ALTER TABLE table_name DROP COLUMN column_name
```

## Parameter

- `table_name` : The name of the table to be modified.
- `column_name` : The name of the column to be deleted.

## Examples

```
ALTER TABLE `TBL` DROP COLUMN `COL2`
```

# ALTER TABLE ADD PARTATION

Last updated: 2026-05-20 18:17:35

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: external tables.
- Purpose: Creates one or more partition columns for a data table.

## Syntax

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
    PARTITION (partition_spec)
    [PARTITION (partition_spec) ...]

partition_spec:
    : partition_column = partition_col_value, partition_column =
partition_col_value, ...
```

## Parameter

- `table_name` : The required table name.
- `partition_column` : The partition column name.
- `partition_col_value` : The partition value.

## Examples

```
ALTER TABLE tbl ADD PARTITION (p1=1, p2='a');

ALTER TABLE tbl ADD IF NOT EXISTS PARTITION (P1 = 1) PARTITION (P2 = 2);
```

# SHOW PARTITIONS

Last updated: 2026-05-20 18:17:52

List all partitions in the table.

Scope: Applies only to Hive external tables, and does not apply to external tables of the Iceberg type or DLC Iceberg internal tables.

## Syntax

```
SHOW PARTITIONS [db_name.]table_name [PARTITION(partition_spec)];
```

## Parameter

- `TABLE [db_name.]table_name` : The table name.
- `[PARTITION(partition_spec)]` : Partition columns. Multiple partition column conditions can be specified.

## Examples

```
SHOW PARTITIONS db01.table PARTITION(ds='2010-03-03', hr='12');
```

# ALTER TABLE DROP PARTITION

Last updated: 2026-05-20 18:18:11

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: external tables.
- Purpose: Deletes a specified partition column from a data table.

## Standard Syntax

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec)
[,PARTITION (partition_spec), ...]
```

partition\_spec:

```
: partition_column = partition_col_value, partition_column =
partition_col_value, ...
```

## Parameter

`table_name` : The required table name.

`partition_column` : The partition name.

`partition_col_value` : The partition value.

## Examples

```
ALTER TABLE page_view DROP PARTITION (dt='2008-08-08', country='us')
```

```
ALTER TABLE `page_view` DROP
PARTITION (`dt` = '2008-08-08', `country` = 'us'),
PARTITION (`dt` = '2008-08-08', `country` = 'us'),
PARTITION (`dt` = '2008-08-08', `country` = 'us')
```

# ALTER TABLE ADD PARTITION FIELD

Last updated: 2026-05-20 18:19:07

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Adds a single partition field to a data table.

## Standard Syntax

```
ALTER TABLE table_name ADD PARTITION partition_column |
hidden_partition_spec [AS alias]

hidden_partition_spec:
    Supported transformations are:
        years(ts): partition by year
        months(ts): partition by month
        days(ts) or date(ts): equivalent to dateint partitioning
        hours(ts) or date_hour(ts): equivalent to dateint and hour
        partitioning
        bucket(N, col): partition by hashed value mod N buckets
        truncate(L, col): partition by value truncated to L
            Strings are truncated to the given length
            Integers and longs truncate to bins: truncate(10, i)
        produces partitions 0, 10, 20, 30, ...
```

## Parameter Description

- `table_name` : The required table name.
- `partition_column` : The partition column.
- `alias` : The alias added to the partition column.

## Examples

```
ALTER TABLE prod.db.sample ADD PARTITION FIELD bucket(16, id)
ALTER TABLE prod.db.sample ADD PARTITION FIELD truncate(data, 4)
```

```
ALTER TABLE prod.db.sample ADD PARTITION FIELD years(ts)
-- use optional AS keyword to specify a custom name for the partition
field
ALTER TABLE prod.db.sample ADD PARTITION FIELD bucket(16, id) AS shard
```



# ALTER TABLE DROP PARTITION FIELD

Last updated: 2026-05-20 18:19:22

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Deletes a specified partition column from a data table.

## Standard Syntax

```
ALTER TABLE table_name ADD PARTITION partition_column |
hidden_partition_spec

hidden_partition_spec:
    Supported transformations are:
        years(ts): partition by year
        months(ts): partition by month
        days(ts) or date(ts): equivalent to dateint partitioning
        hours(ts) or date_hour(ts): equivalent to dateint and hour
        partitioning
        bucket(N, col): partition by hashed value mod N buckets
        truncate(L, col): partition by value truncated to L
            Strings are truncated to the given length
            Integers and longs truncate to bins: truncate(10, i)
        produces partitions 0, 10, 20, 30, ...
```

## Parameter Description

- `table_name` : The required table name.
- `partition_column` : The partition column.

## Examples

```
ALTER TABLE prod.db.sample DROP PARTITION FIELD catalog
ALTER TABLE prod.db.sample DROP PARTITION FIELD bucket(16, id)
ALTER TABLE prod.db.sample DROP PARTITION FIELD truncate(data, 4)
```

```
ALTER TABLE prod.db.sample DROP PARTITION FIELD years(ts)  
ALTER TABLE prod.db.sample DROP PARTITION FIELD shard
```

# ALTER TABLE SET TBLPROPERTIES

Last updated: 2026-05-20 18:19:39

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Updates/deletes data table properties.

## SET: Updating Property Configuration

### Syntax

```
ALTER TABLE table_name
SET TBLPROPERTIES (property_name=property_value, ...)
```

### Parameters

- `table_name` : The required table name.
- `property_name` : The name of the property to be modified.
- `property_value` : The value of the property to be modified.

### Example

```
ALTER TABLE orders SET TBLPROPERTIES ('notes'="Please don't drop this
table.");
```

## UNSET: Deleting Property Configuration

### Syntax

```
ALTER TABLE table_name
UNSET TBLPROPERTIES (property_name, ...)
```

### Parameters

- `table_name` : The required table name.

- `property_name` : The name of the property to be modified.

## Example

```
ALTER TABLE dempts UNSET TBLPROPERTIES ('read.split.target-size')
```

# ALTER TABLE SET LOCATION

Last updated: 2026-05-20 18:19:55

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Modifies the storage path of a data table.

## Standard Syntax

```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new location';
```

## Parameter

- `table_name` : The name of the data table.
- `PARTITION (partition_spec)` : Specifies the partition column(s).
  - `partition_col_name`: The partition column name.
  - `partition_col_value`: The value of the partition column.
- `'new location'` : The location of the new table or partition on tencent cos.

## Examples

```
ALTER TABLE tbl PARTITION (a='1', b='2') SET LOCATION '/path/to/part/ways';

ALTER TABLE tbl SET LOCATION '/path/to/part/ways';
```

# ALTER TABLE ... WRITE ORDERED BY

Last updated: 2026-05-20 18:20:10

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Sets the sorting method for data insertion into tables.

## Syntax

```
ALTER TABLE table_identifier
WRITE [LOCALLY] ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]
```

## Parameter

`table_identifier` : The name of the data table.

## Examples

```
ALTER TABLE dempts WRITE ORDERED BY category, id;
-- use optional ASC/DEC keyword to specify sort order of each field
(default ASC)
ALTER TABLE dempts WRITE ORDERED BY category ASC, id DESC;
-- use optional NULLS FIRST=NULLS LAST keyword to specify null order of
each field (default FIRST)
ALTER TABLE dempts WRITE ORDERED BY category ASC NULLS LAST, id DESC
NULLS FIRST;
-- To order within each task, not across tasks
ALTER TABLE dempts WRITE LOCALLY ORDERED BY category, id;
```

# ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION

Last updated: 2026-05-20 18:20:27

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Modifies the data distribution policy for a partitioned table.

## Syntax

```
ALTER TABLE table_identifier
WRITE DISTRIBUTED BY PARTITION
[ LOCALLY ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]]
```

## Examples

```
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION;
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION LOCALLY ORDERED BY id;
```

# ALTER TABLE ... SET IDENTIFIER FIELDS

Last updated: 2026-05-20 18:20:45

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Adds the identifier fields property.

## Syntax

```
ALTER TABLE dempts SET IDENTIFIER FIELD empno, name
```

## Examples

```
ALTER TABLE tb1 SET IDENTIFIER FIELDS id, location.lon
```



# ALTER TABLE ... DROP IDENTIFIER FIELDS

Last updated: 2026-05-20 18:21:01

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.
- Purpose: Deletes the identifier fields property.

## Syntax

```
ALTER TABLE dempts DROP IDENTIFIER FIELD empno, name
```

## Examples

```
ALTER TABLE tb1 DROP IDENTIFIER FIELDS id, location.lon
```

# MSCK REPAIR TABLE

Last updated: 2026-05-20 18:21:16

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Updates the partition information of a data table.

## Standard Syntax

```
MSCK REPAIR TABLE table_identifier
```

## Parameter

`table_identifier` : The name of the table.

## Examples

```
MSCK REPAIR TABLE t1
```

# ANALYZE TABLES

Last updated: 2026-05-20 18:21:33

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Supports statistics on database tables.
- Limitation: This syntax does not support Iceberg tables.

## Syntax

```
ANALYZE TABLES [ { FROM | IN } database_name ] COMPUTE STATISTICS [
NOSCAN ]

ANALYZE TABLE table_identifier
[ PARTITION ( partition_col_name [ = partition_col_val ] [ , ... ] ) ]
COMPUTE STATISTICS [ NOSCAN | FOR COLUMNS col [ , ... ] | FOR ALL
COLUMNS ]
```

## Parameter

- `database_name` : The database that contains the tables for which statistics need to be calculated.
- `table_identifier` : The name of the table for which statistics need to be calculated.
- `partition_col_name` : The name of the partition column for which statistics need to be calculated.
- `partition_col_value` : The value of the partition column for which statistics need to be calculated.

## Examples

```
ANALYZE TABLE students COMPUTE STATISTICS
ANALYZE TABLE students COMPUTE STATISTICS FOR COLUMNS name
ANALYZE TABLE db.students COMPUTE STATISTICS FOR COLUMNS name
ANALYZE TABLE students COMPUTE STATISTICS NOSCAN
ANALYZE TABLE students COMPUTE STATISTICS FOR all COLUMNS
ANALYZE TABLE db.students COMPUTE STATISTICS FOR all COLUMNS
ANALYZE TABLE students PARTITION (student_id) COMPUTE STATISTICS
```

```
ANALYZE TABLE students PARTITION (student_id = 111111) COMPUTE
STATISTICS
ANALYZE TABLE db.students PARTITION (student_id = 111111, name = 'test')
COMPUTE STATISTICS FOR all COLUMNS

ANALYZE TABLES COMPUTE STATISTICS
ANALYZE TABLES COMPUTE STATISTICS NOSCAN
ANALYZE TABLES from school_db COMPUTE STATISTICS NOSCAN
ANALYZE TABLES IN school_db COMPUTE STATISTICS NOSCAN
```

# DROP TABLE

Last updated: 2026-05-20 18:21:50

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native tables, external tables.
- Purpose: Removes a metadata table.

## Syntax

```
DROP TABLE [IF EXISTS] table_name;
```

## Parameter

- `IF EXISTS` : Optional, meaning "if exists".
- `table_name` : The table name.

## Examples

```
DROP TABLE tbl  
DROP TABLE IF EXISTS tbl
```

# EXPLAIN

Last updated: 2026-05-21 10:44:39

## Description

- Supported engines: Presto, SparkSQL.
- Supported table types: native tables, external tables.
- Purpose: Displays the logical or physical plan for executing sql.

## Syntax

### Presto

```
EXPLAIN [ ( option [, ...] ) ] statement
-- where option can be one of:
-- FORMAT { TEXT | GRAPHVIZ | JSON }
-- TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

### SparkSQL

```
EXPLAIN [ EXTENDED | CODEGEN | COST | FORMATTED ] statement

EXPLAIN ANALYZE
EXPLAIN ANALYZE [VERBOSE] statement
```

## Examples

```
-- presto
EXPLAIN (TYPE VALIDATE) SELECT regionkey, count(*) FROM nation GROUP BY
1;
EXPLAIN (TYPE IO, FORMAT JSON) INSERT INTO test_nation SELECT * FROM
nation WHERE regionkey = 2;

-- EXPLAIN ANALYZE
EXPLAIN ANALYZE SELECT count(*), clerk FROM orders WHERE orderdate >
date '1995-01-01' GROUP BY clerk;
```

```
EXPLAIN ANALYZE VERBOSE SELECT count(clerk) OVER() FROM orders WHERE  
orderdate > date '1995-01-01';
```

# CALL STATEMENT

Last updated: 2026-05-21 09:36:13

The stored procedure is available only when the Iceberg SQL extension is used in Spark.

## Syntax

```
CALL expression
```

## Parameter

`expression` : The function expression.

## Examples

```
CALL catalog_name.`system`.procedure_name(arg_name_2 => arg_2,
arg_name_1 => arg_1)

-- When parameters are passed by position, only the trailing parameters
can be omitted if they are optional.
CALL catalog_name.system.procedure_name(arg_1, arg_2, ... arg_n)

-- Set the current snapshot to db.sample1:
CALL catalog_name.system.set_current_snapshot('db.sample', 1)
```



# CREATE VIEW AS

Last updated: 2026-05-21 10:44:31

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Creates a view.

## Standard Syntax

```
CREATE [ OR REPLACE ] VIEW [IF NOT EXISTS] view_name
    [(column_name [COMMENT 'column_comment'] [, ...])]
    [COMMENT 'view_comment']
AS select_statement
```

## Parameter

- `[IF NOT EXISTS]` : Creates the object if it does not exist.
- `view_name` : The view name.
- `[(column_name [COMMENT 'column_comment'] [, ...])]` : The column name, optionally followed by the column comment.
- `[COMMENT 'view_comment']` : The view comment.
- `select_statement` : The query statement.

## Examples

```
create or replace view db1.v1 as select x,y from tbl;

create view test_view (id comment 'test c1', name_length comment 'test
name c2') as select id, length(name) from test;
```

# SHOW VIEWS

Last updated: 2026-05-21 10:47:13

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Lists the views in a specified database. If you omit the database name, it lists the views in the current database.

## Syntax

```
SHOW VIEWS [IN database_name] LIKE ['regular_expression']
```

## Parameter

- `[IN database_name]` : Specifies the name of the database from which to list the views. If the database name is omitted, the database in the current context is assumed.
- `-` : Filters the view list to show only the views that match the specified regular expression. You can only use the wildcard `*` to represent any character, or to indicate alternatives between characters.

## Examples

```
SHOW VIEWS;
```

```
SHOW VIEWS IN db01 LIKE 'view*';
```

# DESCRIBE VIEW

Last updated: 2026-05-21 10:47:30

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Views view properties.

## Syntax

```
DESCRIBE [view_name]
```

## Parameter

view\_name: The view name.

## Examples

```
DESCRIBE view1;
```

# SHOW CREATE VIEW

Last updated: 2026-05-21 10:47:45

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Displays the statement for creating a view.

## Syntax

```
SHOW CREATE VIEW view_name
```

## Parameter

`view_name` : The view name.

## Examples

```
SHOW CREATE VIEW orders_by_date
```

# SHOW COLUMNS IN VIEW

Last updated: 2026-05-21 10:48:00

## Basic Information

- Supported engines: Presto, SparkSQL.
- Purpose: Views the column information of a view.

## Standard Syntax

```
SHOW COLUMNS IN view_name;
```

## Parameter

`view_name` : The view name.

## Examples

```
SHOW COLUMNS IN view_test
```

# ALTER VIEW

## ALTER VIEW RENAME TO

Last updated: 2026-05-21 11:18:36

### Description

- Supported engines: Presto, SparkSQL.
- Purpose: Changes the view name.

### Standard Syntax

```
ALTER VIEW old_view_identifier RENAME TO new_view_identifier
```

### Parameter

- `old_view_identifier` : The name of the view before modification.
- `new_view_identifier` : The name of the view after modification.

### Examples

```
alter view old_view rename to new_view
```

# ALTER VIEW SET TBLPROPERTIES

Last updated: 2026-05-21 11:53:30

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Modifies view properties.

## Standard Syntax

```
ALTER VIEW view_identifier SET TBLPROPERTIES ( property_key =  
property_val [ , ... ] )
```

## Parameter

- `view_identifier` : The name of the property view to be modified.
- `property_key` : The property name.
- `property_val` : The property value.

## Examples

```
alter view view1 set tblproperties('comment' = 'view1')  
  
alter view view1 set tblproperties('property1' = 'value1', 'property2' =  
'value2')
```

# DROP VIEW

Last updated: 2024-01-12 10:27:13

## Note

- Supported kernels: Presto, SparkSQL.
- Purpose: Deletion of View.

## Syntax

```
DROP VIEW [ IF EXISTS ] view_name;
```

## Category

- `IF EXISTS` : Optional, signifies if it exists.
- `view_name` : Name of the view.

## Sample

```
DROP VIEW orders_by_date;  
DROP VIEW IF EXISTS orders_by_date;
```



# CREATE FUNCTION

Last updated: 2026-05-21 11:55:42

## Description

Supported engines: Presto, SparkSQL.

Purpose: Creates a function implemented by a class name.

## Syntax

```
CREATE FUNCTION [db_name.]function_name AS class_name
  [USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];

-- A function is identified as a UDTF when its suffix is "_udtf".
CREATE FUNCTION [db_name].function_udtf AS class_name
  [USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];
```

## Parameter

- `[db_name.]function_name` : The function name. Specify the namespace under `db_name` when creating the function.
- `class_name` : The implementation class of the function.
- `USING JAR|FILE|ARCHIVE 'file_uri'` : The path to the function resource.

## Examples

```
CREATE FUNCTION `MYFUNC` AS 'myclass' USING JAR 'hdfs:///path/to/jar'

CREATE FUNCTION `MYFUNC` AS 'myclass' USING JAR 'hdfs:///path/to/jar',
FILE 'file:///usr/local/'
```

# SHOW FUNCTION

Last updated: 2026-05-21 14:22:15

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Displays the syntax for creating a function.

## Syntax

```
SHOW FUNCTIONS [ [ LIKE ] { function_name | regex_pattern }
```

## Parameter

- `function_name` : The function name.
- `regex_pattern` : The regular expression used to filter out function names.

## Examples

```
SHOW FUNCTIONS
```

```
SHOW FUNCTIONS trim;
```

```
SHOW FUNCTIONS LIKE 't*'
```

```
SHOW FUNCTIONS LIKE 'yea*|windo*'
```

```
SHOW FUNCTIONS LIKE 't[a-z][a-z][a-z]'
```

# DROP FUNCTION

Last updated: 2026-05-21 14:22:32

## Description

- Supported engines: Presto, SparkSQL.
- Purpose: Deletes a user-defined function.

## Syntax

```
DROP FUNCTION [IF EXISTS] function_name
```

## Parameter

`function_name` : The name of the function to be deleted.

## Examples

```
DROP FUNCTION IF EXISTS `FUNC`
```

```
DROP FUNCTION `FUNC`
```

# DML syntax

## INSERT STATEMENT

Last updated: 2026-05-21 14:22:59

Insert a new row of records into a table. If a column list is specified, it must exactly match the column list generated by the query. Each column in the table that is not in the specified column list will be populated with a null value. If no column list is specified, the columns generated by the query must exactly match the columns in the table being inserted into.

### Syntax

- **Presto:**

```
INSERT INTO table_name [ ( column [, ... ] ) ] query
```

- **Spark:**

```
INSERT INTO table_identifier [ partition_spec ] [ ( column_list ) ]  
{ VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query }
```

### Parameter

- `[ partition_spec ]`: The partition column(s) and value(s). For example, `dt='2021-06-01'`.
- `[ ( column [, ... ] ) ]`: All columns.
- `[table_name] | table_identifier`: The table name.
- `[query]`: A generic Select query statement.

### Examples

General Insert Example for Presto and Spark:

```
INSERT INTO orders SELECT * FROM new_orders;
```

```
INSERT INTO cities VALUES (1, 'China');
```

```
INSERT INTO nation (nationkey, name, regionkey, comment)
```

```
VALUES (26, 'POLAND', 3, 'no comment');
```

### Spark Example:

To insert a partition, use a select query:

```
INSERT INTO students PARTITION (student_id = 444444) SELECT name,  
address FROM persons WHERE name = 'dlc'
```

Insert a partition:

```
INSERT INTO students PARTITION (student_id = 11215017) (address, name)  
VALUES ('Shen zhen, China', 'tester')
```

## Limit

Presto does not support the operation of inserting partitions. If you need to insert a partition, you can use the spark engine to perform the operation.

# INSERT INTO

Last updated: 2026-05-21 14:23:19

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Supports inserting the results of a SELECT query run on a source table as new rows into a target table.

## Syntax

```
[ WITH with_query [ , ... ] ]  
INSERT {INTO [<TABLE>] | TABLE} table_identifier [ partition_spec ] [ (  
column_list) ]  
    { VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query }
```

## Parameter

- `table_identifier` : Specifies the table name. It supports a three-part format, for example, `catalog.database.table`.
- `partition_spec` : The partition column(s) and value(s). For example, `dt='2021-06-01'`.
- `column_list` : All columns.
- `query` : A generic Select query statement.

1.1 a SELECT statement

1.2 a TABLE statement

## Examples

```
INSERT INTO orders SELECT * FROM new_orders;  
INSERT INTO cities VALUES (1, 'China');  
INSERT INTO nation (nationkey, name, regionkey, comment)  
VALUES (26, 'POLAND', 3, 'no comment');  
  
-- INSERT INTO partition  
INSERT INTO students PARTITION (student_id = 444444) SELECT name,  
address FROM persons WHERE name = 'dlc'
```

```
INSERT INTO students PARTITION (student_id = 11215017) (address, name)
VALUES ('Shen zhen, China', 'tester')
```

```
-- Insert Using a TABLE Statement
```

```
INSERT INTO students TABLE visiting_students;
```

```
-- with
```

```
WITH `tmp1` AS ((SELECT *
FROM `catalog1`.`db1`.`tbl1`)), `tmp2` AS ((SELECT *
FROM `tbl2`))
INSERT INTO `catalog1`.`db2`.`tbl1`
(SELECT `col1`, `col2`
FROM `tmp1` `a`
INNER JOIN `tmp2` `b` ON `a`.`col1` = `b`.`col2`)
```

```
INSERT INTO `catalog1`.`db2`.`tbl1`
WITH `tmp1` AS ((SELECT *
FROM `catalog1`.`db1`.`tbl1`)), `tmp2` AS ((SELECT *
FROM `tbl2`))
(SELECT `col1`, `col2`
FROM `tmp1` `a`
INNER JOIN `tmp2` `b` ON `a`.`col1` = `b`.`col2`)
```

# INSERT OVERWRITE

Last updated: 2026-05-21 14:23:36

## Description

- Supported engines: Presto, SparkSQL.
- Applicable table types: native Iceberg tables, external tables.
- Purpose: Row-level data insertion operations.

### Description

Presto supports the insert overwrite operation only on partitioned tables in Hive data sources. This operation is not currently supported for non-partition tables or tables in Iceberg data sources.

## Syntax

```
INSERT OVERWRITE table_identifier [ partition_spec ] [ ( column_list ) ]  
    { VALUES ( { value | NULL } [ , ... ] ) [ , ( ... ) ] | query
```

## Parameter

- `table_identifier` : Specifies the table name. It supports a three-part format, for example, `catalog.database.table`.
- `partition_spec` : The partition column(s) and value(s). For example, `dt='2021-06 -01'`.
- `column_list` : All columns.
- `query` : A generic Select query statement.

1.1 a SELECT statement

1.2 a TABLE statement

## Examples

```
-- Insert Using a VALUES Clause  
INSERT OVERWRITE students VALUES  
    ('Ashua Hill', '456 Erica Ct, Cupertino', 111111),  
    ('Brian Reed', '723 Kern Ave, Palo Alto', 222222);  
  
-- Insert Using a SELECT Statement
```



```
INSERT OVERWRITE students PARTITION (student_id = 222222)
    SELECT name, address FROM persons WHERE name = "Dora Williams"

-- Insert Using a TABLE Statement
INSERT OVERWRITE students TABLE visiting_students

-- Insert with a column list
INSERT OVERWRITE students (address, name, student_id) VALUES
    ('Hangzhou, China', 'Kent Yao', 11215016)

-- Insert with both a partition spec and a column list
INSERT OVERWRITE students PARTITION (student_id = 11215016) (address,
name) VALUES
    ('Hangzhou, China', 'Kent Yao Jr.')
```

# MERGE INTO

Last updated: 2026-05-21 14:23:51

## Description

- Supported engines: SparkSQL.
- Applicable table types: native Iceberg tables.
- Purpose: Row-level data update operations, which can be used to replace INSERT OVERWRITE operations.

## Syntax

```
MERGE INTO tablePrimary1 [ [ AS ] alias ]
USING tablePrimary2
ON booleanExpression
[ WHEN MATCHED (AND matchedCond=booleanExpression)? THEN DELETE ]*
[ WHEN MATCHED (AND matchedCond=booleanExpression)? THEN UPDATE SET
assign [, assign ]* ]*
[ WHEN NOT MATCHED (AND notMatchedCond=booleanExpression)? THEN INSERT
VALUES '(' value [ , value ]*
```

## Parameter

- `tablePrimary1` : Specifies the table name. It supports a three-part format, for example, `catalog.database.table`.
- `alias` : Alias.
- `tablePrimary2` : Can be a table name or a subquery.
- `booleanExpression` : Boolean expression.

## Examples

```
MERGE INTO catalog1.db2.tbl1 t
USING catalog1.db1.tbl1
ON t.col1 = tbl1.col1
WHEN MATCHED AND t.col1 = 14 THEN DELETE

MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1 FROM catalog1.db1.tbl1) s
```

```
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = 14 THEN UPDATE SET col1 = 2

MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1 FROM catalog1.db1.tbl1) s
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = 12 THEN UPDATE SET col1 = 0
WHEN MATCHED AND t.col1 = 13 THEN UPDATE SET col1 = 1
WHEN MATCHED AND t.col1 = 14 THEN UPDATE SET col1 = 2
WHEN MATCHED AND t.col1 = 15 or s.col1 = 16 THEN UPDATE SET col1 =
t.col1 + 1
WHEN MATCHED AND t.col1 not in (12, 13, 14, 15) THEN UPDATE SET col1 = 4
WHEN NOT MATCHED AND t.col1 = 12 THEN INSERT (col1) VALUES (s.col1)
WHEN NOT MATCHED AND t.col1 = 13 THEN INSERT (col1) VALUES (s.col1 + 1)
WHEN NOT MATCHED AND t.col1 = 14 THEN INSERT (col1) VALUES (s.col1 + 2)

MERGE INTO catalog1.db2.tbl1 t
USING (SELECT col1, col2 FROM catalog1.db1.tbl1) s
ON t.col1 = s.col1
WHEN MATCHED AND t.col1 = fun1(s.col2) THEN DELETE
WHEN MATCHED AND t.col1 = db2.fun1(s.col2) THEN DELETE
WHEN MATCHED AND (t.col1 = length(s.col2) or t.col1 =
catalog2.db2.fun3(s.col2)) THEN UPDATE SET col1 = 3
WHEN NOT MATCHED AND t.col1 = 12 THEN INSERT (col1) VALUES
(db2.fun2(s.col2))
```

# UPDATE

Last updated: 2026-05-21 14:24:07

## Description

- Supported engines: SparkSQL.
- Applicable table types: native Iceberg tables.
- Purpose: Updates specified rows in a data table.

## Syntax

```
UPDATE tablePrimary
SET assign [, assign ]*
[ WHERE booleanExpression ]
```

## Parameter

- `tablePrimary` : Specifies the table name. It supports a three-part format, for example, `catalog.database.table`.
- `assign` : The expression to be modified, for example, `col1 = 'new_data'`.
- `booleanExpression` : Boolean expression.

## Examples

```
UPDATE prod.db.table
SET c1 = 'update_c1', c2 = 'update_c2'
WHERE ts >= '2020-05-01 00:00:00' and ts < '2020-06-01 00:00:00'

UPDATE prod.db.all_events
SET session_time = 0, ignored = true
WHERE session_time < (SELECT min(session_time) FROM prod.db.good_events)

UPDATE prod.db.orders AS t1
SET order_status = 'returned'
WHERE EXISTS (SELECT oid FROM prod.db.returned_orders WHERE t1.oid =
oid)
```

# DELETE STATEMENT

Last updated: 2026-05-21 14:24:24

## Description

- Supported engines: SparkSQL.
- Applicable table types: native Iceberg tables.
- Purpose: Deletes specified rows from a data table.

## Syntax

```
DELETE FROM table_name [ [ AS ] alias ]  
[ WHERE booleanExpression ]
```

## Parameter Description

`table_identifier` : Specifies the table name. It supports a three-part format, for example, `catalog.database.table`.

## Examples

```
DELETE FROM lineitem WHERE shipmode = 'AIR';  
  
DELETE FROM lineitem  
WHERE orderkey IN (SELECT orderkey FROM orders WHERE priority = 'LOW');  
  
DELETE FROM orders;
```

# TABLE METADATA

Last updated: 2026-05-21 14:24:42

## Description

- Supported engines: SparkSQL.
- Applicable table types: native Iceberg tables.
- Purpose: Supports four-phase Iceberg table metadata queries, including: history, snapshots, files, and manifests.

## Syntax

```
SELECT select_expr (, select_expr)*  
FROM  
`Catalog`.`db`.`tableName${history|snapshots|files|manifests|partitions|  
all_data_files|all_manifests}`  
[WHERE where_condition]  
[LIMIT [offset,] rows]
```

## Examples

```
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$history` ORDER BY  
snapshot_id DESC LIMIT 1;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$snapshots` ORDER BY  
snapshot_id LIMIT 1;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$files` ORDER BY  
file_size_in_bytes LIMIT 1;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$manifests` ORDER BY  
length LIMIT 1;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$partitions` LIMIT  
10;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_data_files`  
LIMIT 10;  
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_manifests`  
LIMIT 10;
```

# DQL syntax

## SELECT STATEMENT

Last updated: 2024-01-12 10:28:33

**SELECT** Statement: Retrieves data rows from zero or more tables.

## Syntax

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...]
]
[ LIMIT [ count | ALL ] ]
```

## Category

### [ WITH with\_query [, ....]]

The WITH clause can be used to flatten nested queries or simplify subqueries. The syntax for `with_query` is as follows:

```
subquery_table_name [ ( column_name [, ...] ) ] AS (subquery)
```

- The `subquery_table_name` is the unique name of the temporary table that defines the result of the WITH clause subquery. Each subquery must have a table name that can be referenced in the FROM clause.
- The `column_name [, ...]` is an optional list of output column names. The number of column names must be equal to or less than the number of columns defined by the subquery.
- A subquery is any query statement.

### [ ALL | DISTINCT ] select\_expr

The ALL and DISTINCT options specify whether duplicate rows should be returned. If these options are not provided, the default is ALL (return all matching rows). DISTINCT specifies the

removal of duplicate rows from the result set.

## FROM from\_item [, ...]

The from\_item can be a view, table, or subquery. If multiple tables are joined, the supported join types are as follows:

- [ INNER ] JOIN
- LEFT [ OUTER ] JOIN
- RIGHT [ OUTER ] JOIN
- FULL [ OUTER ] JOIN
- CROSS JOIN
- ON join\_condition , if join\_condition is used, you can specify column names for join keys in multiple tables; if join\_column is used, it requires that join\_column exists in both tables.

## [ WHERE condition ]

The results are filtered based on the condition you specify, returning a result set that meets the criteria.

## [ GROUP BY [ ALL | DISTINCT ] grouping\_expressions [, ...] ]

The GROUP BY expression allows for the grouping of output according to specified column names.

## [HAVING condition ]

Used in conjunction with aggregate functions and the GROUP BY clause. It controls which groups are selected, thereby eliminating groups that do not meet the condition. This filtering occurs after the calculation of groups and aggregation.

## [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] union\_query]

UNION , INTERSECT , and EXCEPT combine multiple results together, with UNION combining the rows generated by the first query with those generated by the second. To eliminate duplicates, UNION constructs a hash table, which consumes memory. For better performance, it is recommended to use UNION ALL.

- INTERSECT only returns the rows that exist in the results of both the first and second queries.
- EXCEPT returns the rows in the first query result, excluding the rows found in the second query.

## [ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]



The result set is sorted by one or more output expressions. When the clause contains multiple expressions, the result set is sorted according to the first `expression`. Subsequently, the second `expression` is applied to rows with matching values in the first expression, and so on.

## Sample

### WITH Clause

The WITH clause defines named relations used in the query. It allows for the flattening of nested queries or the simplification of subqueries. For instance, the following queries are equivalent:

```
WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)
SELECT a, b FROM x;
```

It can also follow multiple subqueries:

```
WITH
  t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),
  t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)
SELECT t1.*, t2.*
FROM t1
JOIN t2 ON t1.a = t2.a;
```

### GROUP BY Clause

The GROUPBY clause divides the output of the SELECT statement into groups of rows with matching values. A simple GROUPBY clause can contain any expression composed of input columns, or it can be an ordinal that selects output columns by position:

```
SELECT count(*), nationkey FROM customer GROUP BY 2;
```

```
SELECT count(*), nationkey FROM customer GROUP BY nationkey;
```

```
SELECT count(*) FROM customer GROUP BY mktsegment;
```

### GROUPING SETS

Grouping sets allow users to specify multiple lists of columns to be grouped. Columns not belonging to the given sub-list of grouping columns are set to null.

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));
```

```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state
UNION ALL
SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip
UNION ALL
SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

## HAVING Clause

The HAVING clause is used in conjunction with aggregate functions and the GROUP BY clause to control which groups are selected. The HAVING clause eliminates groups that do not meet the given conditions.

```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
```

## IN

The IN operator allows you to specify multiple values in the WHERE clause.

```
SELECT name
FROM nation
```

```
WHERE regionkey IN (SELECT regionkey FROM region)
```

## EXISTS

The EXISTS operator is used to determine whether there are records in the query clause. If one or more records exist, it returns True; otherwise, it returns False.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition)
```

## USING

Simplified using the 'using' keyword.

- The query must be an equijoin.
- In an equijoin, the columns must have identical names and data types.

```
SELECT *
FROM table_1
JOIN table_2
USING (key_A, key_B)
```

```
SELECT *
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)
LEFT JOIN (
    VALUES
        (1, 3, 100),
        (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
USING (key_A, key_B);
```

## CROSS JOIN

A cross join returns the Cartesian product of two relations (all combinations).

```
SELECT *  
FROM nation  
CROSS JOIN region
```

## LIMIT Clause

The LIMIT clause restricts the number of rows in the result set.

```
SELECT orderdate FROM orders LIMIT 5
```

## ORDER BY Clause

The ORDER BY clause is used to sort the result set based on one or more output expressions.

```
Syntax: ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [,  
...]
```

```
SELECT name, age FROM person ORDER BY age
```

```
SELECT * FROM student  
ORDER BY student_id
```

```
SELECT * FROM student  
ORDER BY student_id, student_name
```

## EXCEPT

The EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means that EXCEPT only returns rows that are not available in the second SELECT statement. The same rules apply when using the EXCEPT operator, as with the UNION operation.

```
SELECT * FROM (VALUES 13, 42)  
EXCEPT  
SELECT 13
```

## INTERSECT

Returns distinct rows from two or more result sets from the SELECT statement.

```
SELECT * FROM (VALUES 13, 42)
INTERSECT
SELECT 1
```

## UNION

Combine the result sets of two or more SELECT statements into one result set. To retain duplicate rows in the result set, use the `UNION ALL` operator.

```
SELECT 13
UNION
SELECT 42
```

```
SELECT id FROM a
UNION ALL
SELECT id FROM b;
```

## TABLESAMPLE

- **BERNOULLI** : Selects each row as a table sample with a probability equal to the sample percentage. When sampling a table using the Bernoulli method, all physical blocks of the table are scanned and certain rows are skipped (based on a comparison between the sampling percentage and a randomly calculated value at runtime). The probability of a row being included in the results is independent of any other row. This does not reduce the time required to read the sample table from the disk. Further processing of the sample output may affect the total query time.
- **SYSTEM** : This sampling method divides the table into logical data segments and samples the table at this granularity. This sampling method either selects all rows from a specific data segment or skips it (based on the comparison between the sampling percentage and the randomly calculated value at runtime). The rows selected in system sampling will depend on the connector used. For example, when used with Hive, it depends on how the data is laid out on HDFS. This method cannot guarantee independent sampling probabilities.

```
SELECT *
```

```
FROM users TABLESAMPLE BERNOULLI (50);
```

```
SELECT *  
FROM users TABLESAMPLE SYSTEM (75);
```

## PIVOT Clause

Returns aggregated values based on specific columns.

```
SELECT * FROM person  
  PIVOT (  
    SUM(age) AS a, AVG(class) AS c  
    FOR name IN ('John' AS john, 'Mike' AS mike)  
  );
```

## Lateral View Clause

```
LATERAL VIEW [ OUTER ] generator_function ( expression [ , ... ] ) [  
table_alias ] AS column_alias [ , ... ]
```

## Escaping

To escape a single quote, precede it with another single quote, as demonstrated in the following example:

```
Select 'dlc''test'
```

Specify an escape character when creating a table, for instance, by using the following method:

```
CREATE EXTERNAL TABLE IF NOT EXISTS csv_test_2222 (  
  _c0 STRING,  
  _c1 INTEGER,  
  _c2 INTEGER,  
  _c3 INTEGER  
)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH  
SERDEPROPERTIES (
```

```
'separatorChar' = ',',  
'quoteChar' = '''  
)  
STORED AS textfile  
LOCATION 'cosn://dlc-nj-1258469122/csv/100M/'
```

# Iceberg Table Syntax

## DDL Syntax

Last updated: 2026-05-21 14:28:30

### ⓘ Note:

The following syntax descriptions apply to DLC native tables, which are Iceberg tables by default. If you use an Iceberg external table, there are detailed differences in DDL syntax. Refer to the document [Iceberg External Table vs. Native Table Syntax Differences](#) for details.

## CREATE TABLE

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

### Parameters

#### table\_identifier

`table_identifier` supports a three-part format: `catalog.db.name` .

### Schemas and Data Types

```
col_type
: primitive_type
  | nested_type

primitive_type
: boolean
  | int/integer
  | long/bigint
  | float
  | double
```



```
| decimal(p,s), where p is the maximum number of digits, s is the
maximum number of decimal places, and s <= p <= 38.
| date
| timestamp and timestamp with timezone are supported, while time and
without timezone are not supported.
| string, which can also correspond to the Iceberg uuid type.
| binary, which can also correspond to the Iceberg fixed type.

nested_type
: struct
| list
| map
```

## Partition Transforms

```
transform
: identity, which supports any type. DLC does not support this
conversion.
| bucket[N], which uses hash mod N for bucketing, supports the following
column types: int, long, decimal, date, timestamp, string, binary.
| truncate[L], which performs bucketing by truncating L, supports the
following column types: int, long, decimal, string.
| years, which represents the year, supports the following column types:
date, timestamp.
| months, which represents the month, supports the following column
types: date, timestamp.
| days/date, which represents the day, supports the following column
types: date, timestamp.
| hours/date_hour, which represents the hour, supports the following
column type: timestamp.
```

## Example

```
CREATE TABLE dempts(
  id bigint COMMENT 'id number',
  num int,
  eno float,
  dno double,
  cno decimal(9,3),
```

```
flag boolean,  
data string,  
ts_year timestamp,  
date_month date,  
bno binary,  
point struct<x: double, y: double>,  
points array<struct<x: double, y: double>>,  
pointmaps map<struct<x: int>, struct<a: int>>  
)  
COMMENT 'table documentation'  
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month), identity(bno), bucket(3,num),truncate(10,data));
```

## CREATE TABLE AS SELECT

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier  
[ COMMENT table_comment ]  
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]  
[ TBLPROPERTIES ( property_name=property_value, ... ) ]  
AS select_statement
```

### Example

```
CREATE TABLE dempts_copy  
COMMENT 'table create as select'  
PARTITIONED BY (eno, dno)  
AS SELECT * from dempts;
```

## REPLACE TABLE AS SELECT

Update the table by creating a Snapshot from the result of a SELECT query while preserving the table History.

### Syntax

```
CREATE [OR REPLACE] TABLE table_identifier  
[ COMMENT table_comment ]
```

```
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]  
AS select_statement
```

## Example

```
CREATE OR REPLACE TABLE dempts_replace  
  COMMENT 'table create as replace'  
  PARTITIONED BY (eno, dno)  
AS SELECT * from dempts;
```

## DROP TABLE

### Syntax

```
DROP TABLE [ IF EXISTS ] table_identifier
```

## ALTER TABLE

Alter table syntax.

### ALTER TABLE ... RENAME TO

Rename a table.

### Syntax

```
ALTER TABLE table_identifier RENAME TO new_table_identifier
```

## ALTER TABLE ... SET / UNSET TBLPROPERTIES

Updates/deletes data table properties.

### Syntax

```
-- SET: Updates property configurations.  
ALTER TABLE table_identifier  
SET TBLPROPERTIES (property_name=property_value, ...)  
  
-- UNSET: Deletes property configurations.  
ALTER TABLE table_identifier
```

```
UNSET TBLPROPERTIES (property_name, ...)
```

## Example

```
-- SET: Updates property configurations.
ALTER TABLE dempts SET TBLPROPERTIES ('read.split.target-
size'='268435456');

-- UNSET: Deletes property configurations.
ALTER TABLE dempts UNSET TBLPROPERTIES ('read.split.target-
size'='268435456');
```

## ALTER TABLE ... WRITE ORDERED BY

Sets the sorting method for data insertion into tables.

### Syntax

```
ALTER TABLE table_identifier
WRITE [LOCALLY] ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]
```

## Example

```
ALTER TABLE dempts WRITE ORDERED BY category, id;

-- Use the optional ASC/DESC keywords to specify the sort order for each
field (ASC by default).
ALTER TABLE dempts WRITE ORDERED BY category ASC, id DESC;

-- Use the optional NULLS FIRST=NULLS LAST keywords to specify the null
ordering for each field (FIRST by default).
ALTER TABLE dempts WRITE ORDERED BY category ASC NULLS LAST, id DESC
NULLS FIRST;

-- Sort within each task, not across tasks.
ALTER TABLE dempts WRITE LOCALLY ORDERED BY category, id;
```

## ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION

Modifies the data distribution policy for a partitioned table.

### Syntax

```
ALTER TABLE table_identifier
WRITE DISTRIBUTED BY PARTITION
[ LOCALLY ORDERED BY
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]]
```

### Example

```
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION;
ALTER TABLE dempts WRITE DISTRIBUTED BY PARTITION LOCALLY ORDERED BY id;
```

## ALTER TABLE COLUMNS

Changes the field syntax.

## ALTER TABLE ... ADD COLUMNS

Adds multiple fields.

### Syntax

```
-- Adds multiple fields.
ALTER TABLE table_identifier
ADD COLUMNS (col_name col_type [COMMENT col_comment], ...)

-- Adds a single field.
ALTER TABLE table_identifier
ADD COLUMN col_name col_type [COMMENT col_comment]
[FIRST | AFTER target_col_name]
```

### Example

```
-- Adds multiple fields.
ALTER TABLE dempts
ADD COLUMNS (
    new_column_1 string comment 'new_column_1 docs',
```

```
    new_column_2 int comment 'new_column_2 docs'
);

-- Adds a single field.
ALTER TABLE dempts
ADD COLUMN new_column_3 string comment 'new_column docs';
```

## ALTER TABLE ... RENAME COLUMN

Changes the field name.

### Syntax

```
ALTER TABLE table_identifier
RENAME COLUMN old_column_name TO new_column_name
```

## ALTER TABLE ... ALTER COLUMN

Changes the field type/remarks.

### Syntax

```
ALTER TABLE table_identifier
ALTER COLUMN col_name
{TYPE new_col_type | COMMENT col_comment}
```

Currently, Iceberg TYPE changes only support safe field type extensions:

- int/integer -> long/bigint
- float -> double
- decimal(P,S) -> decimal(P2,S), where  $P2 > P$ , meaning the precision is increased.

### Example

```
ALTER TABLE dempts ALTER COLUMN new_column_2 TYPE bigint;
ALTER TABLE dempts ALTER COLUMN new_column_2 comment 'alter docs';
```

## ALTER TABLE ... DROP COLUMN

Deletes a table field.

### Syntax

```
ALTER TABLE table_identifier DROP COLUMN column_name
```

## ALTER TABLE PARTITIONS

### ALTER TABLE ... ADD PARTITION FIELD

Adds a single partition field.

#### Syntax

```
ALTER TABLE table_identifier  
ADD PARTITION FIELD col_name|transform (col_name) [AS alias]
```

For transform, refer to the [CREATE TABLE](#) description.

#### Example

```
ALTER TABLE dempts ADD PARTITION FIELD new_column_1;  
ALTER TABLE dempts ADD PARTITION FIELD bucket(3,new_column_2);
```

### ALTER TABLE ... REPLACE PARTITION FIELD

Replaces a single partition field.

#### Syntax

```
ALTER TABLE table_identifier REPLACE PARTITION FIELD col_name|transform  
(col_name) [AS alias]
```

### ALTER TABLE ... DROP PARTITION FIELD

Deletes a single partition field.

#### Syntax

```
ALTER TABLE table_identifier  
DROP PARTITION FIELD col_name|transform (col_name);
```

For transform, refer to the [CREATE TABLE](#) description.

#### Example

```
ALTER TABLE dempts DROP PARTITION FIELD new_column_1;  
ALTER TABLE dempts DROP PARTITION FIELD bucket(3,new_column_2);
```

## ALTER TABLE ... SET IDENTIFIER FIELDS

Adds the identifier fields property.

### Syntax

```
ALTER TABLE dempts SET IDENTIFIER FIELDS empno, name
```

## ALTER TABLE .. DROP IDENTIFIER FIELDS

Adds the identifier fields property.

### Syntax

```
ALTER TABLE dempts DROP IDENTIFIER FIELDS empno, name
```



# DML Syntax

Last updated: 2026-05-21 14:28:53

## INSERT OVERWRITE | INTO

Row-level data insertion operations

### Syntax

```
INSERT { OVERWRITE | INTO } [ TABLE ] table_name
[ PARTITION clause ]
{ VALUES (column_values,...), (column_values,...)...
| SELECT select_expr}
```

### Example

```
CREATE TABLE IF NOT EXISTS `table_01` (
  `id` INTEGER,
  `num` int,
  `name` STRING
) USING `iceberg`

INSERT INTO table_01 PARTITION(name='21') VALUES (1,2), (2,3);
INSERT INTO TABLE table_01 VALUES (3,2,'abc'), (4,3,'abd');
```

## MERGE INTO

Row-level data update operations can be used to replace INSERT OVERWRITE operations.

### Syntax

```
MERGE INTO target_table_name [target_alias]
USING source_table_reference [source_alias]
ON merge_condition
[ WHEN MATCHED [ AND condition ] THEN matched_action ] [...]
[ WHEN NOT MATCHED [ AND condition ] THEN not_matched_action ] [...]

matched_action
{ DELETE |
```

```
UPDATE SET * |
UPDATE SET { column1 = value1 } [, ...] }

not_matched_action
{ INSERT * |
INSERT (column1 [, ...] ) VALUES (value1 [, ...])
```

## DELETE FROM

### Syntax

```
DELETE FROM table_name [table_alias] [WHERE predicate]
```

## UPDATE

UPDATE operations are supported from Spark 3.1 onward.

### Syntax

```
UPDATE table_identifier [table_alias]
SET { { column_name | field_name } = expr } [, ...]
[WHERE clause]
```

## Example

```
UPDATE dempts SET c1 = 'update_c1', c2 = 'update_c2'
WHERE ts >= '2020-05-01 00:00:00' and ts < '2020-06-01 00:00:00'

UPDATE dempts SET session_time = 0, ignored = true
WHERE session_time < (SELECT min(session_time) FROM prod.db.good_events)

UPDATE dempts AS t1 SET order_status = 'returned'
WHERE EXISTS (SELECT oid FROM prod.db.returned_orders WHERE t1.oid =
oid)
```



# DQL Syntax

Last updated: 2026-05-21 14:29:11

## SELECT

### Syntax

```
SELECT [ hints ] [ ALL | DISTINCT ] { named_expression | star_clause }
[, ...]
FROM from_item [, ...]
    [ LATERAL VIEW clause ]
    [ PIVOT clause ]
    [ WHERE clause ]
    [ GROUP BY clause ]
    [ HAVING clause ]
    [ QUALIFY clause ]

from_item
{ table_name [ TABLESAMPLE clause ] [ table_alias ] |
  JOIN clause |
  [ LATERAL ] table_valued_function [ table_alias ] |
  VALUES clause |
  [ LATERAL ] ( query ) [ TABLESAMPLE clause ] [ table_alias ] }

named_expression
  expression [ column_alias ]

star_clause
  [ { table_name | view_name } . ] *
```

## TABLE METADATA

Supports four-phase Iceberg table metadata queries, including: `history` , `snapshots` , `files` , and `manifests` .

### Syntax

```
SELECT select_expr (, select_expr)*
```

```
FROM
`Catalog`.`db`.`tableName${history|snapshots|files|manifests|partitions|
all_data_files|all_manifests}`
[WHERE where_condition]
[LIMIT [offset,] rows]
```

## Example

```
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$history` ORDER BY
snapshot_id DESC LIMIT 1;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$snapshots` ORDER BY
snapshot_id LIMIT 1;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$files` ORDER BY
file_size_in_bytes LIMIT 1;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$manifests` ORDER BY
length LIMIT 1;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$partitions` LIMIT
10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_data_files`
LIMIT 10;
SELECT * FROM `DataLakeCatalog`.`validation`.`dempts$all_manifests`
LIMIT 10;
```

## TIME TRAVEL

### FOR SYSTEM\_TIME AS OF/ TIMESTAMP AS OF

Supported from Spark 3.3 onward, it accommodates both string and Unix timestamp formats.

#### Example

```
SELECT empno FROM sales.emp FOR SYSTEM_TIME AS OF '1986-10-26 01:21:00';
SELECT empno FROM sales.emp FOR SYSTEM_TIME AS OF 12324235546;
SELECT empno FROM sales.emp TIMESTAMP AS OF '1986-10-26 01:21:00';
SELECT empno FROM sales.emp TIMESTAMP AS OF 11111;
```

### FOR SYSTEM\_VERSION AS OF/ VERSION AS OF (Supported in Spark 3.3)

Supports strings and snapshot IDs.

## Example

```
SELECT empno FROM sales.emp VERSION AS OF 'Snapshot123456789';  
SELECT empno FROM sales.emp VERSION AS OF 11111;  
SELECT empno FROM sales.emp FOR SYSTEM_VERSION AS OF  
'Snapshot123456789';  
SELECT empno FROM sales.emp FOR SYSTEM_VERSION AS OF 11111;
```

## AT\_TIMESTAMP\_XXXX

### Example

```
SELECT * FROM `sales`.`emp$at_timestamp_1111`;
```

## SNAPSHOT\_ID\_XXXX

### Example

```
SELECT * FROM `sales`.`emp$snapshot_id_1111`
```

# Procedure

Last updated: 2026-05-21 14:29:27

## Description

- Supported engines: SparkSQL.
- Applicable table types: external Iceberg tables, native Iceberg tables.

## Basic Syntax

```
CALL catalog_name.system.procedure_name(arg_name_2 => arg_2, arg_name_1
=> arg_1);

CALL catalog_name.system.procedure_name(arg_1, arg_2, ... arg_n);
```

## Snapshot Management

### rollback\_to\_snapshot

Roll back a snapshot to a specified version.

Input parameters: table name and version number.

```
CALL
`DataLakeCatalog`.`system`.rollback_to_snapshot('validation.dempts', 1);
```

### rollback\_to\_timestamp

Roll back a snapshot to a specified timestamp.

Input parameters: table name and timestamp.

```
CALL
`DataLakeCatalog`.`system`.rollback_to_timestamp('validation.dempts',
TIMESTAMP '2022-08-11 19:49:43.224');
```

### set\_current\_snapshot

Set the current snapshot version.

Input parameters: table name and version number.

```
CALL
`DataLakeCatalog`.`system`.set_current_snapshot('validation.dempts', 1);
```

## cherrypick\_snapshot

Cherry-pick from a specified snapshot version to the current snapshot.

```
CALL `DataLakeCatalog`.`system`.cherrypick_snapshot('validation.dempts',
1);
CALL `DataLakeCatalog`.`system`.cherrypick_snapshot(snapshot_id => 1,
table => 'my_table' )
```

## Metadata Management

### expire\_snapshots

Clean up expired snapshots to reduce the number of small files.

```
CALL `Catalog`.`system`.expire_snapshots(table_name, [older_than],
[retain_last], [max_concurrent_deletes], [stream_results]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.expire_snapshots('validation.dempts',
TIMESTAMP '2021-06-30 00:00:00.000', 100);
```

### remove\_orphan\_files

Remove metadata files that are no longer referenced.

```
CALL `Catalog`.`system`.remove_orphan_files(table_name, [older_than],
[location], [dry_run], [max_concurrent_deletes]);
```

Example:

```
CALL
`DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.demp
ts', dry_run=>TRUE);
```



```
CALL
`DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.demp
ts', `location`=>'cosn://channingdata-1305424723/example2/');
CALL `DataLakeCatalog`.`system`.remove_orphan_files('validation.dempts',
TIMESTAMP '2022-07-10 17:25:19.000');
```

## remove\_orphan\_files

Remove metadata files that are no longer referenced.

```
CALL `Catalog`.`system`.remove_orphan_files(table_name, [older_than],
[location], [dry_run], [max_concurrent_deletes]);
```

Example:

```
CALL
`DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.demp
ts', dry_run=>TRUE);
CALL
`DataLakeCatalog`.`system`.remove_orphan_files(`table`=>'validation.demp
ts', `location`=>'cosn://channingdata-1305424723/example2/');
CALL `DataLakeCatalog`.`system`.remove_orphan_files('validation.dempts',
TIMESTAMP '2022-07-10 17:25:19.000');
```

## rewrite\_data\_files

Data files are merged and rewritten, which means small data files are consolidated.

```
CALL `Catalog`.`system`.rewrite_data_files(table_name, [strategy],
[sort_order], [options], [where]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.rewrite_data_files('validation.dempts');
CALL
`DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempt
s', `strategy`=>'sort', `sort_order`=>'id DESC NULLS LAST,data ASC NULLS
FIRST');
```

```
CALL
`DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempt
s', `options`=>map('min-input-files','2'));
CALL
`DataLakeCatalog`.`system`.rewrite_data_files(`table`=>'validation.dempt
s', `where`=>'id = 3 and data = "foo"');
```

## rewrite\_manifests

Manifest files are merged and rewritten.

```
CALL `Catalog`.`system`.rewrite_manifests(table_name, [using_caching]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.rewrite_manifests('validation.dempt');
CALL `DataLakeCatalog`.`system`.rewrite_manifests('validation.dempt',
FALSE);
```

## ancestors\_of

Obtain the lineage information of the snapshot.

```
CALL `Catalog`.`system`.ancestors_of(table_name, [snapshot_id]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.ancestors_of('validation.dempt');
CALL `DataLakeCatalog`.`system`.ancestors_of('validation.dempt', 1);
```

## Data Table Migration Management

### Note:

The source table must be a Hive table or a Spark table.

## snapshot

Create a lightweight temporary table based on the source table, and the snapshot of the source table is directly reused for the temporary table.

```
CALL `Catalog`.`system`.snapshot(source_table, table, [location],  
[properties]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01',  
'validation.snap');  
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01',  
'validation.snap2', 'cosn://channingdata-1305424723/example3/');
```

## migrate

Update or replace table properties.

```
CALL `Catalog`.`system`.migrate(table, [properties]);
```

Example:

```
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01');  
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01',  
map('data', 'name'));
```

## add\_files

Load data files directly from hive, and you can specify the target partition for the data files.

```
CALL `Catalog`.`system`.add_files(table, source_table,  
[partition_filter]);
```

Example:

```
CALL  
`DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02',  
`source_table`=>'validation.table_01');  
CALL  
`DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02',
```

```
`source_table`=>'validation.table_01',  
`partition_filter`=>map('part_col', 'A'));
```

# Syntax Differences Between Iceberg External Tables and Native Tables

Last updated: 2026-05-21 14:29:47

DLC uses Iceberg syntax version 0.13.1. For detailed syntax specifications, refer to [Iceberg Official Documentation](#).

When you use an Iceberg external table, the SQL syntax differs from that of a native Iceberg table in the following ways.

## CREATE TABLE

### Native Tables

#### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
[ COMMENT table_comment ]
[ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
```

#### Examples

```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>>
)
COMMENT 'table documentation'
```

```
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month),
identity(bno), bucket(3,num), truncate(10,data));
```

## External Tables

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    ( col_name[:] col_type [ COMMENT col_comment ], ... )
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
```

### Examples

```
CREATE TABLE dempts(
    id bigint COMMENT 'id number',
    num int,
    eno float,
    dno double,
    cno decimal(9,3),
    flag boolean,
    data string,
    ts_year timestamp,
    date_month date,
    bno binary,
    point struct<x: double, y: double>,
    points array<struct<x: double, y: double>>,
    pointmaps map<struct<x: int>, struct<a: int>>
)
USING iceberg
COMMENT 'table documentation'
PARTITIONED BY (bucket(16,id), years(ts_year), months(date_month),
identity(bno), bucket(3,num), truncate(10,data))
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
TBLPROPERTIES ('write.format.default'='orc');
```

# CREATE TABLE AS SELECT

## Native Tables

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
AS select_statement
```

### Examples

```
CREATE TABLE IF NOT EXISTS dempts_copy
COMMENT 'table create as select'
PARTITIONED BY (eno, dno)
AS SELECT * from dempts;
```

## External Tables

### Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

### Examples

```
CREATE TABLE dempts_copy
USING iceberg
COMMENT 'table create as select'
PARTITIONED BY (eno, dno)
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
TBLPROPERTIES ('write.format.default'='avro')
AS SELECT * from dempts;
```

# REPLACE TABLE AS SELECT

## Native Tables

### Syntax

```
CREATE OR REPLACE TABLE table_identifier
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
AS select_statement
```

### Examples

```
CREATE OR REPLACE TABLE dempts_replace
COMMENT 'table create as replace'
PARTITIONED BY (eno, bucket(10, num))
AS SELECT * from dempts;
```

## External Tables

### Syntax

```
CREATE [OR REPLACE] TABLE table_identifier
USING iceberg
    [ COMMENT table_comment ]
    [ PARTITIONED BY ( col_name1, transform(col_name2), ... ) ]
    [ LOCATION path ]
    [ TBLPROPERTIES ( property_name=property_value, ... ) ]
AS select_statement
```

### Examples

```
CREATE OR REPLACE TABLE dempts_replace
USING iceberg
COMMENT 'table create as replace'
PARTITIONED BY (eno, dno)
LOCATION 'cosn://rickytest-1305424723/channing-test/loc'
```



```
TBLPROPERTIES ('write.format.default'='avro')
AS SELECT * from dempts;
```

## Procedure

### Note:

The source table to be migrated must be a Hive table or a Spark table.

## Native Tables

Not supported.

## External Tables

- **snapshot**

Creates a lightweight temporary table based on the source table. The temporary table directly reuses the source table snapshot.

### Syntax

```
CALL `Catalog`.`system`.snapshot(source_table, table, [location],
[properties]);
```

### Examples

```
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01',
'validation.snap');
CALL `DataLakeCatalog`.`system`.snapshot('validation.table_01',
'validation.snap2', 'cosn://channingdata-1305424723/example3/');
```

- **call**

Updates or replaces table properties.

### Syntax

```
CALL `Catalog`.`system`.migrate(table, [properties]);
```

### Examples

```
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01');
```

```
CALL `DataLakeCatalog`.`system`.migrate('validation.table_01',  
map('data', 'name'));
```

- **add\_files**

Loads data files directly from hive. You can specify the target partition for the data files.

### Syntax

```
CALL `Catalog`.`system`.add_files(table, source_table,  
[partition_filter]);
```

### Examples

```
CALL  
`DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02',  
`source_table`=>'validation.table_01');  
CALL  
`DataLakeCatalog`.`system`.add_files(`table`=>'validation.table_02',  
`source_table`=>'validation.table_01',  
`partition_filter`=>map('part_col', 'A'));
```

# SQL Implicit Conversion

Last updated: 2026-05-21 14:30:03

To facilitate SQL writing and enable a quick switch from Hive/Spark syntax to the unified SQL in DLC, we provide SQL implicit cast capability. The conversion list is as follows:  
I: implicit cast, E: explicit cast, --: not allowed

| From-To     | null | boolean | tinyint | smallint | int | bigint | decimal | float/real | double | interval | date | time | timestamp | [var]char | [var]binary |
|-------------|------|---------|---------|----------|-----|--------|---------|------------|--------|----------|------|------|-----------|-----------|-------------|
| null        | I    | I       | I       | I        | I   | I      | I       | I          | I      | I        | I    | I    | I         | I         | I           |
| boolean     | --   | I       | E       | E        | E   | E      | E       | E          | E      | --       | --   | --   | --        | I         | --          |
| tinyint     | --   | E       | I       | I        | I   | I      | I       | I          | I      | E        | --   | --   | E         | I         | --          |
| smallint    | --   | E       | I       | I        | I   | I      | I       | I          | I      | E        | --   | --   | E         | I         | --          |
| int         | --   | E       | I       | I        | I   | I      | I       | I          | I      | E        | --   | --   | E         | I         | --          |
| bigint      | --   | E       | I       | I        | I   | I      | I       | I          | I      | E        | --   | --   | E         | I         | --          |
| decimal     | --   | E       | I       | I        | I   | I      | I       | I          | I      | E        | --   | --   | E         | I         | --          |
| float/real  | --   | E       | I       | I        | I   | I      | I       | I          | I      | --       | --   | --   | E         | I         | --          |
| double      | --   | E       | I       | I        | I   | I      | I       | I          | I      | --       | --   | --   | E         | I         | --          |
| interval    | --   | --      | E       | E        | E   | E      | E       | --         | --     | I        | --   | --   | --        | E         | --          |
| date        | --   | --      | --      | --       | --  | --     | --      | --         | --     | --       | I    | --   | I         | I         | --          |
| time        | --   | --      | --      | --       | --  | --     | --      | --         | --     | --       | --   | I    | E         | I         | --          |
| timestamp   | --   | --      | E       | E        | E   | E      | E       | E          | E      | --       | I    | E    | I         | I         | --          |
| [var]char   | --   | E       | I       | I        | I   | I      | I       | I          | I      | I        | I    | I    | I         | I         | I           |
| [var]binary | --   | --      | --      | --       | --  | --     | --      | --         | --     | --       | E    | E    | E         | I         | I           |

# Function

## Unified Function

## Unified Function Overview

Last updated: 2026-05-21 14:30:33

DLC supports using unified functions across different kernels and is compatible with both Spark and Presto. For specific functions and supported kernels, refer to the table below. DLC also supports Presto built-in functions. For the list of supported functions and how to enable them, see [Presto Built-in Functions](#).

| Function              | Spark                                                                    | Presto                                                   |
|-----------------------|--------------------------------------------------------------------------|----------------------------------------------------------|
| ABS                   | Input parameters: bigint double decimal                                  | Input parameters: bigint double decimal                  |
|                       | Output parameters: Same as input parameters                              | Output parameters: double                                |
| ACOS                  | Input parameters: bigint double decimal                                  | Input parameters: bigint double decimal                  |
|                       | Output parameters: double                                                | Output parameters: double                                |
| ACOSH                 | Input parameters: bigint double decimal                                  | Input parameters: bigint double decimal                  |
|                       | Output parameters: double                                                | Output parameters: double                                |
| ADD_MONTHS            | Input parameters: date timestamp string, int                             | Input parameters: date timestamp string, int             |
|                       | Output parameters: date                                                  | Output parameters: date                                  |
| AES_DECRYPT           | –                                                                        | Input parameters: (binary, string binary)                |
|                       |                                                                          | Output parameters: binary                                |
| AES_ENCRYPT           | –                                                                        | Input parameters: (string binary, string binary)         |
|                       |                                                                          | Output parameters: binary                                |
| AND                   | Input parameters: boolean AND boolean                                    | boolean AND boolean                                      |
|                       | Output parameters: boolean                                               | Output parameters: boolean                               |
| ANY                   | Input parameters: boolean                                                | Input parameters: boolean                                |
|                       | Output parameters: boolean                                               | Output parameters: boolean                               |
| ANY_MATCH             | Input parameters: (array<T>, lambda function)                            | –                                                        |
|                       | Output parameters: boolean                                               |                                                          |
| APPROX_COUNT_DISTINCT | Input parameters: (bigint double decimal date timestamp,)                | –                                                        |
|                       | (double array<double>[, bigint])                                         |                                                          |
|                       | Output parameters: Same as the second input parameter                    |                                                          |
| APPROX_PERCENTILE     | Input parameters: (bigint double decimal[, array<double> double[, int]]) | Input parameters: (bigint double decimal, double[, int]) |
|                       | Output parameters: double array                                          | Output parameters: double array                          |
| ARG_MAX               | Input parameters: (col1, col2   expr(col2))                              | –                                                        |
|                       | Output parameters: the type of col2 or expr(col2)                        |                                                          |
| ARG_MIN               | Input parameters: (col1, col2   expr(col2))                              |                                                          |
|                       | Output parameters: the type of col2 or expr(col2)                        |                                                          |
| ARRAY                 | Input parameters: (T, ...)                                               | Input parameters: (T, ...)                               |
|                       | Output parameters: array<T>                                              | Output parameters: array<T>                              |
| ARRAYS_OVERLAP        | Input parameters: (array<T>, array<T>)                                   | Input parameters: (array<T>, array<T>)                   |
|                       | Output parameters: boolean                                               | Output parameters: boolean                               |
| ARRAYS_ZIP            | Input parameters: (array<T>, array<U>, ...)                              | –                                                        |
|                       | Output parameters: array<struct<T, U, ...>>                              |                                                          |
| ARRAY_CONTAINS        | Input parameters: (array<T>, T)                                          | Input parameters: (array<T>, T)                          |
|                       | Output parameters: boolean                                               | Output parameters: boolean                               |
| ARRAY_DISTINCT        | Input parameters: array<T>                                               | Input parameters: array<T>                               |
|                       | Output parameters: array<T>                                              | Output parameters: array<T>                              |
| ARRAY_EXCEPT          | Input parameters: (array<T>, array<T>)                                   | Input parameters: (array<T>, array<T>)                   |
|                       | Output parameters: array<T>                                              | Output parameters: array<T>                              |
| ARRAY_INTERSECT       | Input parameters: (array<T>, array<T>)                                   | Input parameters: (array<T>, array<T>)                   |
|                       | Output parameters: array<T>                                              | Output parameters: array<T>                              |
| ARRAY_JOIN            | Input parameters: (array<T>, string[, string])                           | Input parameters: (array<T>, string[, string])           |
|                       | Output parameters: string                                                | Output parameters: string                                |
| -----                 | Input parameters: array<bigint double decimal>                           | Input parameters: array<bigint double decimal>           |

|                |                                                                  |                                                                  |
|----------------|------------------------------------------------------------------|------------------------------------------------------------------|
| ARRAY_MAX      | Output parameters: bigint double decimal                         | Output parameters: bigint double decimal                         |
|                |                                                                  |                                                                  |
| ARRAY_MIN      | Input parameters: array<bigint double decimal>                   | Input parameters: array<bigint double decimal>                   |
|                | Output parameters: bigint double decimal                         | Output parameters: bigint double decimal                         |
| ARRAY_POSITION | Input parameters: (array<T>, bigint)                             | Input parameters: (array<T>, bigint)                             |
|                | Output parameters: bigint                                        | Output parameters: bigint                                        |
| ARRAY_REMOVE   | Input parameters: (array<T>, T)                                  | Input parameters: (array<T>, T)                                  |
|                | Output parameters: array<T>                                      | Output parameters: array<T>                                      |
| ARRAY_REPEAT   | Input parameters: (array<T>, bigint)                             | Input parameters: (array<T>, bigint)                             |
|                | Output parameters: array<T>                                      | Output parameters: array<T>                                      |
| ARRAY_SORT     | Input parameters: (array<T>, function(T, T)->integer)            | Input parameters: (array<T>)                                     |
|                | Output parameters: array<T>                                      | Output parameters: array<T>                                      |
| ARRAY_UNION    | Input parameters: (array<T>, array<T>)                           | Input parameters: (array<T>, array<T>)                           |
|                | Output parameters: array<T>                                      | Output parameters: array<T>                                      |
| ASCII          | Input parameters: string                                         | Input parameters: string                                         |
|                | Output parameters: int                                           | Output parameters: int                                           |
| ASIN           | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                        | Output parameters: double                                        |
| ASINH          | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                        | Output parameters: double                                        |
| ASSERT_TRUE    | Input parameters: boolean                                        | Input parameters: boolean                                        |
|                | Output parameters: null or throws an exception                   | Output parameters: null or throws an exception                   |
| ATAN           | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                        | Output parameters: double                                        |
| ATAN2          | Input parameters: (bigint double decimal, bigint double decimal) | Input parameters: (bigint double decimal, bigint double decimal) |
|                | Output parameters: double                                        | Output parameters: double                                        |
| ATANH          | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                        | Output parameters: double                                        |
| AVG            | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                        | Output parameters: double                                        |
| BASE64         | Input parameters: string binary                                  | Input parameters: binary                                         |
|                | Output parameters: string                                        | Output parameters: string                                        |
| BIGINT         | Casts to bigint                                                  | Casts to bigint                                                  |
| BIN            | Input parameters: bigint                                         | Input parameters: bigint                                         |
|                | Output parameters: string                                        | Output parameters: string                                        |
| BINARY         | Casts to binary                                                  | Casts to binary                                                  |
| BIT_AND        | Input parameters: int                                            | Input parameters: int                                            |
|                | Output parameters: int                                           | Output parameters: int                                           |
| BIT_COUNT      | Input parameters: int boolean                                    | Input parameters: int boolean                                    |
|                | Output parameters: int                                           | Output parameters: int                                           |
| BIT_GET        | Input parameters: (int, int)                                     | Input parameters: (int, int)                                     |
|                | Output parameters: int                                           | Output parameters: int                                           |
| BIT_LENGTH     | Input parameters: string                                         | Input parameters: string                                         |
|                | Output parameters: int                                           | Output parameters: int                                           |
| BIT_OR         | Input parameters: int                                            | Input parameters: int                                            |
|                | Output parameters: int                                           | Output parameters: int                                           |

|                  |                                                            |                                                                   |
|------------------|------------------------------------------------------------|-------------------------------------------------------------------|
| BIT_XOR          | Input parameters: int                                      | Input parameters: int                                             |
|                  | Output parameters: int                                     | Output parameters: int                                            |
| BOOLEAN          | Casts to boolean.                                          | Casts to boolean.                                                 |
| BOOL_AND         | Input parameters: boolean                                  | Input parameters: boolean                                         |
|                  | Output parameters: boolean                                 | Output parameters: boolean                                        |
| BOOL_OR          | Input parameters: boolean                                  | Input parameters: boolean                                         |
|                  | Output parameters: boolean                                 | Output parameters: boolean                                        |
| BROUND           | Input parameters: (bigint double decimal, int)             | Input parameters: (bigint double decimal, int)                    |
|                  | Output parameters: int                                     | Output parameters: int                                            |
| BTRIM            | Input parameters: (string[, string])                       | Input parameters: (string[, string])                              |
|                  | Output parameters: string                                  | Output parameters: string                                         |
| CARDINALITY      | Input parameters: (array map)                              | Input parameters: (array map)                                     |
|                  | Output parameters: int                                     | Output parameters: int                                            |
| CAST             | Input parameters: (<expr> AS T)                            | Input parameters: (<expr> AS T)                                   |
|                  | Output parameters: T                                       | Output parameters: T                                              |
| CBRT             | Input parameters: bigint double decimal                    | Input parameters: bigint double decimal                           |
|                  | Output parameters: double                                  | Output parameters: double                                         |
| CEIL             | Input parameters: bigint double decimal                    | Input parameters: bigint double decimal                           |
|                  | Output parameters: bigint                                  | Output parameters: bigint                                         |
| CEILING          | Input parameters: bigint double decimal                    | Input parameters: bigint double decimal                           |
|                  | Output parameters: bigint                                  | Output parameters: bigint                                         |
| CHAR             | Input parameters: int                                      | Input parameters: int                                             |
|                  | Output parameters: string                                  | Output parameters: string                                         |
| CHARACTER_LENGTH | Input parameters: string binary                            | Input parameters: string                                          |
|                  | Output parameters: int                                     | Output parameters: int                                            |
| CHAR_LENGTH      | Input parameters: string binary                            | Input parameters: string                                          |
|                  | Output parameters: int                                     | Output parameters: int                                            |
| CHR              | Input parameters: int double                               | Input parameters: int double                                      |
|                  | Output parameters: string                                  | Output parameters: string                                         |
| CLUSTER_SAMPLE   | Input parameters: (bigint [, bigint])                      | –                                                                 |
|                  | Output parameters: boolean                                 |                                                                   |
| COALESCE         | Input parameters: (T, T, ...)                              | Input parameters: (T, T, ...)                                     |
|                  | Output parameters: T                                       | Output parameters: T                                              |
| COLLECT_LIST     | Input parameters: T                                        | Input parameters: T                                               |
|                  | Output parameters: array<T>                                | Output parameters: array<T>                                       |
| COLLECT_SET      | Input parameters: T                                        | Input parameters: T                                               |
|                  | Output parameters: array<T>                                | Output parameters: array<T>                                       |
| CONCAT           | Input parameters: string array                             | Input parameters: (string binary, string binary, ...)             |
|                  | Output parameters: Same as the input parameter             | Output parameters: string                                         |
| CONCAT_WS        | Input parameters: (string, [string   array<string>]+)      | Input parameters: (string, [string   array<string>]+)             |
|                  | Output parameters: string                                  | Output parameters: string                                         |
| CONTEXT_NGRAMS   | –                                                          | Input parameters: (array<array<string>>, array<string>, int, int) |
|                  |                                                            | Output parameters: array<struct<string, double>>                  |
| CONV             | Input parameters: (bigint double decimal string, int, int) | Input parameters: (bigint double decimal string, int, int)        |
|                  | Output parameters: string                                  | Output parameters: string                                         |

|                   | Input parameters: string                                         | Output parameters: string                                        |
|-------------------|------------------------------------------------------------------|------------------------------------------------------------------|
| CORR              | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
|                   | Output parameters: double                                        |                                                                  |
| COS               | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| COSH              | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| COT               | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| COUNT             | Input parameters: T                                              | Input parameters: T                                              |
|                   | Output parameters: bigint                                        | Output parameters: bigint                                        |
| COUNT_IF          | Input parameters: boolean                                        | -                                                                |
|                   | Output parameters: int                                           |                                                                  |
| COUNT_MIN_SKETCH  | Input parameters: (int binary string, double, double, int)       | -                                                                |
|                   | Output parameters: binary                                        |                                                                  |
| COVAR_POP         | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| COVAR_SAMP        | Input parameters: bigint double decimal                          | Input parameters: bigint double decimal                          |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| CRC32             | Input parameters: string binary                                  | Input parameters: string binary                                  |
|                   | Output parameters: bigint                                        | Output parameters: bigint                                        |
| CUME_DIST         | Input parameters: None                                           | Input parameters: None                                           |
|                   | Output parameters: double                                        | Output parameters: double                                        |
| CURRENT_CATALOG   | Input parameters: None                                           | -                                                                |
|                   | Output parameters: string                                        |                                                                  |
| CURRENT_DATABASE  | Input parameters: None                                           | -                                                                |
|                   | Output parameters: string                                        |                                                                  |
| CURRENT_DATE      | Input parameters: None                                           | Input parameters: None                                           |
|                   | Output parameters: date                                          | Output parameters: date                                          |
| CURRENT_TIMESTAMP | Input parameters: None                                           | Input parameters: None                                           |
|                   | Output parameters: timestamp                                     | Output parameters: timestamp                                     |
| CURRENT_TIMEZONE  | Input parameters: None                                           | Input parameters: None                                           |
|                   | Output parameters: string                                        | Output parameters: string                                        |
| CURRENT_USER      | Input parameters: None                                           | Input parameters: None                                           |
|                   | Output parameters: string                                        | Output parameters: string                                        |
| DATE              | Input parameters: string                                         | Input parameters: string                                         |
|                   | Output parameters: date                                          | Output parameters: date                                          |
| DATEDIFF          | Input parameters: (string timestamp date, string timestamp date) | Input parameters: (string timestamp date, string timestamp date) |
|                   | Output parameters: int                                           | Output parameters: int                                           |
| DATE_ADD          | Input parameters: (string date timestamp, int)                   | Input parameters: (string date timestamp, int)                   |
|                   | Output parameters: date                                          | Output parameters: date                                          |
| DATEADD           | Input parameters: (date timestamp, int, string)                  | -                                                                |
|                   | Output parameters: timestamp                                     |                                                                  |
| DATE_FORMAT       | Input parameters: (string date timestamp, string)                | Input parameters: (string date timestamp, string)                |
|                   | Output parameters: string                                        | Output parameters: string                                        |



|                     |                                                                                      |                                                    |
|---------------------|--------------------------------------------------------------------------------------|----------------------------------------------------|
|                     | Output parameters: string                                                            | Output parameters: string                          |
| DATE_FROM_UNIX_DATE | Input parameters: int                                                                | Input parameters: int                              |
|                     | Output parameters: date                                                              | Output parameters: date                            |
| DATE_PART           | Input parameters: (string, string date timestamp)                                    | Input parameters: (string, string date timestamp)  |
|                     | Output parameters: bigint double decimal                                             | Output parameters: bigint double decimal           |
| DATE_SUB            | Input parameters: (string date timestamp)                                            | Input parameters: (string date timestamp)          |
|                     | Output parameters: date                                                              | Output parameters: date                            |
| DATE_TRUNC          | Input parameters: (string, string)                                                   | Input parameters: (string, string)                 |
|                     | Output parameters: timestamp                                                         | Output parameters: timestamp                       |
| DAY                 | Input parameters: string timestamp date                                              | Input parameters: string timestamp date            |
|                     | Output parameters: int                                                               | Output parameters: int                             |
| DAYOFMONTH          | Input parameters: string timestamp date                                              | Input parameters: string timestamp date            |
|                     | Output parameters: int                                                               | Output parameters: int                             |
| DAYOFWEEK           | Input parameters: string timestamp date                                              | string timestamp date                              |
|                     | Output parameters: int                                                               | Output parameters: int                             |
| DAYOFYEAR           | Input parameters: string timestamp date                                              | string timestamp date                              |
|                     | Output parameters: int                                                               | Output parameters: int                             |
| DECIMAL             | cast as decimal                                                                      | cast as decimal                                    |
| DECODE              | Input parameters: (binary, string)   (int, int, string[, int, string]...[, default]) | Input parameters: (binary, string)                 |
|                     | Output parameters:                                                                   | Output parameters: string                          |
| DEGREES             | Input parameters: bigint double decimal                                              | Input parameters: bigint double decimal            |
|                     | Output parameters: double                                                            | Output parameters: double                          |
| DENSE_RANK          | Input parameters: None                                                               | Input parameters: None                             |
|                     | Output parameters: int                                                               | Output parameters: int                             |
| DIV                 | Input parameters: bigint double decimal                                              | -                                                  |
|                     | Output parameters: int                                                               |                                                    |
| DOUBLE              | cast as double                                                                       | cast as double                                     |
| ELEMENT_AT          | Input parameters: (array map, int)                                                   | Input parameters: (array, int)                     |
|                     | Output parameters: Same as the array element type or map value type.                 | Output parameters: Same as the array element type. |
| ELT                 | Input parameters: (int, T, U,...)                                                    | Input parameters: (int, string, string, ...)       |
|                     | Output parameters: Depends on the input.                                             | Output parameters: string                          |
| ENCODE              | Input parameters: (string binary, string)                                            | Input parameters: (string, string)                 |
|                     | Output parameters: binary                                                            | Output parameters: binary                          |
| EVERY               | Input parameters: T                                                                  | Input parameters: T                                |
|                     | Output parameters: boolean                                                           | Output parameters: boolean                         |
| EXP                 | Input parameters: bigint double decimal                                              | Input parameters: bigint double decimal            |
|                     | Output parameters: double                                                            | Output parameters: double                          |
| EXPLODE             | Input parameters: array<T>                                                           | -                                                  |
|                     | Output parameters: T                                                                 |                                                    |
| EXPLODE_OUTER       | Input parameters: array<T>                                                           | -                                                  |
|                     | Output parameters: T                                                                 |                                                    |
| EXPM1               | Input parameters: bigint double decimal                                              | -                                                  |
|                     | Output parameters: double                                                            |                                                    |
| EXTRACT             | Input parameters: field FROM source                                                  | Input parameters: field FROM source                |
|                     | Output parameters: The return type varies depending on the field                     | Output parameters: bigint                          |

|                     | type.                                                                 | Output parameters: bigint                                               |
|---------------------|-----------------------------------------------------------------------|-------------------------------------------------------------------------|
| FACTORIAL           | Input parameters: int                                                 | Input parameters: int                                                   |
|                     | Output parameters: bigint                                             | Output parameters: bigint                                               |
| FIELD               | –                                                                     | Input parameters: (T, T, T, ...)                                        |
|                     |                                                                       | Output parameters: int                                                  |
| FIND_IN_SET         | Input parameters: (string, string)                                    | Input parameters: (string, string)                                      |
|                     | Output parameters: int                                                | Output parameters: int                                                  |
| FIRST_VALUE         | Input parameters: (T[, boolean])                                      | –                                                                       |
|                     | Output parameters: T                                                  |                                                                         |
| FLATTEN             | Input parameters: array<array<T>>                                     | Input parameters: array<array<T>>                                       |
|                     | Output parameters: array<T>                                           | Output parameters: array<T>                                             |
| FLOAT               | cast as float                                                         | cast as float                                                           |
| FLOOR               | Input parameters: bigint double decimal                               | Input parameters: bigint double decimal                                 |
|                     | Output parameters: bigint                                             | Output parameters: bigint                                               |
| FORMAT_NUMBER       | Input parameters: (bigint double decimal, int string)                 | Input parameters: (bigint double decimal, int)                          |
|                     | Output parameters: string                                             | Output parameters: string                                               |
| FORMAT_STRING       | Input parameters: (string, T, ...)                                    | Input parameters: (string, T, ...)                                      |
|                     | Output parameters: string                                             | Output parameters: string                                               |
| FROM_CSV            | Input parameters: (string, string[, map])                             | –                                                                       |
|                     | Output parameters: struct                                             |                                                                         |
| FROM_JSON           | Input parameters: (string, string[, map])                             | –                                                                       |
|                     | Output parameters: struct                                             |                                                                         |
| FROM_UNIXTIME       | Input parameters: (bigint[, string])                                  | Input parameters: (bigint[, string])                                    |
|                     | Output parameters: string                                             | Output parameters: string                                               |
| FROM_UTC_TIMESTAMP  | Input parameters: (string, string)                                    | Input parameters: (bigint double decimal timestamp date string, string) |
|                     | Output parameters: timestamp                                          | Output parameters: timestamp                                            |
| GETBIT              | Input parameters: (int, int)                                          | Input parameters: (int, int)                                            |
|                     | Output parameters: int                                                | Output parameters: int                                                  |
| GET_IDCARD_AGE      | Input parameters: string                                              | –                                                                       |
|                     | Output parameters: int                                                |                                                                         |
| GET_IDCARD_BIRTHDAY | Input parameters: string                                              | –                                                                       |
|                     | Output parameters: date                                               |                                                                         |
| GET_IDCARD_SEX      | Input parameters: string                                              | –                                                                       |
|                     | Output parameters: string                                             |                                                                         |
| GREATEST            | Input parameters: (bigint double decimal, bigint double decimal, ...) | Input parameters: (bigint double decimal, bigint double decimal, ...)   |
|                     | Output parameters: Same as the input parameter                        | Output parameters: Same as the input parameter                          |
| GROUPING            | Input parameters: T                                                   | T                                                                       |
|                     | Output parameters: int                                                | Output parameters: int                                                  |
| GROUPING_ID         | Input parameters: [T[, T...]]                                         | –                                                                       |
|                     | Output parameters: int                                                |                                                                         |
| HASH                | Input parameters: (T, T, ...)                                         | Input parameters: (T, T, ...)                                           |
|                     | Output parameters: int                                                | Output parameters: int                                                  |
| HEX                 | Input parameters: int string binary                                   | Input parameters: int string binary                                     |
|                     | Output parameters: string                                             | Output parameters: string                                               |
|                     |                                                                       | Input parameters: bigint double decimal                                 |

|                         |                                                       |                                            |
|-------------------------|-------------------------------------------------------|--------------------------------------------|
| HISTOGRAM_NUMERIC       | -                                                     | Input parameters: bigint double decimal    |
|                         |                                                       | Output parameters: array<struct<'x', 'y'>> |
| HOUR                    | Input parameters: string timestamp                    | Input parameters: string timestamp         |
|                         | Output parameters: int                                | Output parameters: int                     |
| HYPOT                   | Input parameters: bigint double decimal               | Input parameters: bigint double decimal    |
|                         | Output parameters: double                             | Output parameters: double                  |
| IF                      | Input parameters: (boolean, T, T)                     | Input parameters: (boolean, T, T)          |
|                         | Output parameters: T                                  | Output parameters: T                       |
| IFNULL                  | Input parameters: (T, U)                              | -                                          |
|                         | Output parameters: T U                                |                                            |
| IN                      | Input parameters: (T, T, ...)                         | Input parameters: (T, T, ...)              |
|                         | Output parameters: boolean                            | Output parameters: boolean                 |
| INITCAP                 | Input parameters: string                              | Input parameters: string                   |
|                         | Output parameters: string                             | Output parameters: string                  |
| INLINE                  | Input parameters: array                               | -                                          |
|                         | Output parameters: table                              |                                            |
| INLINE_OUTER            | Input parameters: array                               | -                                          |
|                         | Output parameters: table                              |                                            |
| INPUT_FILE_BLOCK_LENGTH | Input parameters: None                                | -                                          |
|                         | Output parameters: int                                |                                            |
| INPUT_FILE_BLOCK_START  | Input parameters: None                                | -                                          |
|                         | Output parameters: int                                |                                            |
| INPUT_FILE_NAME         | Input parameters: None                                | -                                          |
|                         | Output parameters: string                             |                                            |
| INSTR                   | Input parameters: (string, string)                    | Input parameters: (string, string)         |
|                         | Output parameters: int                                | Output parameters: int                     |
| INT                     | Casts to int                                          | Casts to int.                              |
| ISNAN                   | Input parameters: T                                   | Input parameters: T                        |
|                         | Output parameters: boolean                            | Output parameters: boolean                 |
| ISNOTNULL               | Input parameters: T                                   | Input parameters: T                        |
|                         | Output parameters: boolean                            | Output parameters: boolean                 |
| ISNULL                  | Input parameters: T                                   | Input parameters: T                        |
|                         | Output parameters: boolean                            | Output parameters: boolean                 |
| JSON_ARRAY_LENGTH       | Input parameters: string                              | Input parameters: string                   |
|                         | Output parameters: int                                | Output parameters: int                     |
| JSON_OBJECT_KEYS        | Input parameters: string                              | Input parameters: string                   |
|                         | Output parameters: array                              | Output parameters: array                   |
| JSON_TUPLE              | Input parameters: (string, ..., string)               | -                                          |
|                         | Output parameters: string...string                    |                                            |
| KEYVALUE                | Input parameters: (string [, string, string], string) |                                            |
|                         | Output parameters: string                             |                                            |
| KURTOSIS                | Input parameters: bigint double decimal               | -                                          |
|                         | Output parameters: double                             |                                            |
| LAG                     | Input parameters: (T[, int[, string]])                | Input parameters: (T[, int[, string]])     |
|                         | Output parameters: T                                  | Output parameters: T                       |
| LAST DAY                | Input parameters: string date timestamp               | Input parameters: string date timestamp    |

|                |                                                                       |                                                                  |
|----------------|-----------------------------------------------------------------------|------------------------------------------------------------------|
| LAST_DAY       | Output parameters: string                                             | Output parameters: string                                        |
| LAST_VALUE     | Input parameters: (T[, boolean])                                      | –                                                                |
|                | Output parameters: T                                                  |                                                                  |
| LCASE          | Input parameters: string                                              | Input parameters: string                                         |
|                | Output parameters: string                                             | Output parameters: string                                        |
| LEAD           | Input parameters: (T[, int[, string]])                                | Input parameters: (T[, int[, string]])                           |
|                | Output parameters: T                                                  | Output parameters: T                                             |
| LEAST          | Input parameters: (bigint double decimal, bigint double decimal, ...) | (bigint double decimal, bigint double decimal, ...)              |
|                | Output parameters: Same as the input parameter                        | Output parameters: Same as the input parameter                   |
| LEFT           | Input parameters: (string, int)                                       | Input parameters: (string, int)                                  |
|                | Output parameters: string                                             | Output parameters: string                                        |
| LENGTH         | Input parameters: string                                              | Input parameters: string                                         |
|                | Output parameters: int                                                | Output parameters: int                                           |
| LEVENSHTEIN    | Input parameters: (string, string)                                    | Input parameters: (string, string)                               |
|                | Output parameters: int                                                | Output parameters: int                                           |
| LIKE           | Input parameters: (string, string)   string LIKE string ESCAPE string | Input parameters: (string, string)                               |
|                | Output parameters: boolean                                            | Output parameters: boolean                                       |
| LN             | Input parameters: bigint double decimal                               | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                             | Output parameters: double                                        |
| LOCATE         | Input parameters: (string, string[, int])                             | Input parameters: (string, string[, int])                        |
|                | Output parameters: int                                                | Output parameters: int                                           |
| LOG            | Input parameters: (bigint double decimal, bigint double decimal)      | Input parameters: (bigint double decimal, bigint double decimal) |
|                | Output parameters: double                                             | Output parameters: double                                        |
| LOG10          | Input parameters: bigint double decimal                               | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                             | Output parameters: double                                        |
| LOG1P          | Input parameters: bigint double decimal                               | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                             | Output parameters: double                                        |
| LOG2           | Input parameters: bigint double decimal                               | Input parameters: bigint double decimal                          |
|                | Output parameters: double                                             | Output parameters: double                                        |
| LOWER          | Input parameters: string                                              | Input parameters: string                                         |
|                | Output parameters: string                                             | Output parameters: string                                        |
| LPAD           | Input parameters: (string, int[, string])                             | Input parameters: (string, int, string)                          |
|                | Output parameters: string                                             | Output parameters: string                                        |
| LTRIM          | Input parameters: string                                              | Input parameters: string                                         |
|                | Output parameters: string                                             | Output parameters: string                                        |
| MAKE_DATE      | Input parameters: (int, int, int)                                     | Input parameters: (int, int, int)                                |
|                | Output parameters: date                                               | Output parameters: date                                          |
| MAKE_TIMESTAMP | Input parameters: (int, int, int, int, int, int[, string])            | Input parameters: (int, int, int, int, int, int[, string])       |
|                | Output parameters: timestamp                                          | Output parameters: timestamp                                     |
| MAP            | Input parameters: (K1, V1, K2, V2, ...)                               | Input parameters: (array<K>, array<V>)                           |
|                | Output parameters: map                                                | Output parameters: map                                           |
| MAP_CONCAT     | Input parameters: (map, ...)                                          | Input parameters: (map, ...)                                     |
|                | Output parameters: map                                                | Output parameters: map                                           |
| MAP_ENTRIES    | Input parameters: map                                                 | Input parameters: map                                            |
|                | Output parameters: array                                              | Output parameters: array                                         |

|                             |                                                                                                                    |                                                                  |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| MAP_FROM_ARRAYS             | Input parameters: (array, array)                                                                                   | Input parameters: (array, array)                                 |
|                             | Output parameters: map                                                                                             | Output parameters: map                                           |
| MAP_FROM_ENTRIES            | Input parameters: array                                                                                            | Input parameters: array                                          |
|                             | Output parameters: map                                                                                             | Output parameters: map                                           |
| MAP_KEYS                    | Input parameters: map                                                                                              | Input parameters: map                                            |
|                             | Output parameters: array                                                                                           | Output parameters: array                                         |
| MAP_UNION_SUM               | Input parameters: map                                                                                              | –                                                                |
|                             | Output parameters: map                                                                                             |                                                                  |
| MAP_VALUES                  | Input parameters: map                                                                                              | Input parameters: map                                            |
|                             | Output parameters: array                                                                                           | Output parameters: array                                         |
| MAX                         | Input parameters: bigint double decimal                                                                            | Input parameters: bigint double decimal                          |
|                             | Output parameters: Same as input parameters                                                                        | Output parameters: double                                        |
| MAX_BY                      | Input parameters: T, bigint double decimal                                                                         | Input parameters: T, bigint double decimal                       |
|                             | Output parameters: T                                                                                               | Output parameters: T                                             |
| MAX_PT                      | Input parameters: const string                                                                                     | –                                                                |
|                             | Output parameters: string                                                                                          |                                                                  |
| MD5                         | Input parameters: string binary                                                                                    | Input parameters: string binary                                  |
|                             | Output parameters: string                                                                                          | Output parameters: string                                        |
| MEAN                        | Input parameters: bigint double decimal                                                                            | Input parameters: bigint double decimal                          |
|                             | Output parameters: double                                                                                          | Output parameters: double                                        |
| MIN                         | Input parameters: bigint double decimal                                                                            | Input parameters: bigint double decimal                          |
|                             | Output parameters: Same as input parameters                                                                        | Output parameters: double                                        |
| MINUTE                      | Input parameters: (string timestamp)                                                                               | Input parameters: (string timestamp)                             |
|                             | Output parameters: int                                                                                             | Output parameters: int                                           |
| MIN_BY                      | Input parameters: (T, bigint double decimal)                                                                       | Input parameters: (T, bigint double decimal)                     |
|                             | Output parameters: T                                                                                               | Output parameters: T                                             |
| MOD                         | Input parameters: (bigint double decimal, bigint double decimal)   bigint double decimal MOD bigint double decimal | Input parameters: (bigint double decimal, bigint double decimal) |
|                             | Output parameters: int double                                                                                      | Output parameters: int double                                    |
| MONOTONICALLY_INCREASING_ID | Input parameters: None                                                                                             | –                                                                |
|                             | Output parameters: bigint                                                                                          |                                                                  |
| MONTH                       | Input parameters: date timestamp string                                                                            | Input parameters: date timestamp string                          |
|                             | Output parameters: int                                                                                             | Output parameters: int                                           |
| MONTHS_BETWEEN              | Input parameters: (date timestamp string, date timestamp string[, boolean])                                        | Input parameters: (date timestamp string, date timestamp string) |
|                             | Output parameters: double                                                                                          | Output parameters: double                                        |
| NAMED_STRUCT                | Input parameters: (K1, V1, K2, V2, ...)                                                                            | –                                                                |
|                             | Output parameters: struct                                                                                          |                                                                  |
| NANVL                       | Input parameters: (bigint double decimal, bigint double decimal)                                                   | Input parameters: (bigint double decimal, bigint double decimal) |
|                             | Output parameters: double                                                                                          | Output parameters: double                                        |
| NEGATIVE                    | Input parameters: bigint double decimal                                                                            | Input parameters: bigint double decimal                          |
|                             | Output parameters: bigint double decimal                                                                           | Output parameters: int double                                    |
| NEXT_DAY                    | Input parameters: (string date timestamp, string)                                                                  | Input parameters: (string date timestamp, string)                |
|                             | Output parameters: date                                                                                            | Output parameters: string                                        |
| NGRAMS                      | –                                                                                                                  | Input parameters: (array<array<string>>, int, int, int)          |
|                             |                                                                                                                    | Output parameters: array<struct<string, double>>                 |

|                   |                                                                                 |                                                                             |
|-------------------|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
|                   |                                                                                 | Output parameters: array   struct   string, double   ...                    |
| NOW               | Input parameters: None                                                          | Input parameters: None                                                      |
|                   | Output parameters: timestamp                                                    | Output parameters: timestamp                                                |
| NTH_VALUE         | Input parameters: (T[, int])                                                    | Input parameters: (T[, int])                                                |
|                   | Output parameters: T                                                            | Output parameters: T                                                        |
| NTILE             | Input parameters: int                                                           | Input parameters: int                                                       |
|                   | Output parameters: int                                                          | Output parameters: int                                                      |
| NULLIF            | Input parameters: (T, T)                                                        | Input parameters: (T, T)                                                    |
|                   | Output parameters: T                                                            | Output parameters: T                                                        |
| NVL               | Input parameters: (T, T)                                                        | Input parameters: (T, T)                                                    |
|                   | Output parameters: T                                                            | Output parameters: T                                                        |
| NVL2              | Input parameters: (T, T, T)                                                     | Input parameters: (T, T, T)                                                 |
|                   | Output parameters: T                                                            | Output parameters: T                                                        |
| OCTET_LENGTH      | Input parameters: string                                                        | Input parameters: string                                                    |
|                   | Output parameters: int                                                          | Output parameters: int                                                      |
| OR                | Input parameters: expr1 OR expr2                                                | Input parameters: expr1 OR expr2                                            |
|                   | Output parameters: boolean                                                      | Output parameters: boolean                                                  |
| OVERLAY           | Input parameters: (string PLACING string FROM int[ FOR int])                    | –                                                                           |
|                   | Output parameters:                                                              |                                                                             |
| PARSE_URL         | Input parameters: (string, string[, string])                                    | Input parameters: (string, string[, string])                                |
|                   | Output parameters: string                                                       | Output parameters: string                                                   |
| PERCENTILE        | Input parameters: (bigint double decimal, bigint double decimal array, int)     | Input parameters: (bigint double decimal, bigint double decimal array, int) |
|                   | Output parameters: bigint double decimal array                                  | Output parameters: double array                                             |
| PERCENTILE_APPROX | Input parameters: (bigint double decimal, bigint double decimal   array[, int]) | Input parameters: (bigint double decimal, bigint double decimal[, int])     |
|                   | Output parameters: double array                                                 | Output parameters: double array                                             |
| PERCENT_RANK      | Input parameters: None                                                          | Input parameters: None                                                      |
|                   | Output parameters: double                                                       | Output parameters: double                                                   |
| PI                | Input parameters: None                                                          | Input parameters: None                                                      |
|                   | Output parameters: double                                                       | Output parameters: double                                                   |
| PMOD              | Input parameters: (bigint double decimal, bigint double decimal)                | Input parameters: (bigint double decimal, bigint double decimal)            |
|                   | Output parameters: Same as input parameters                                     | Output parameters: Same as input parameters                                 |
| POSITION          | Input parameters: (string, string[, int])   (string IN string)                  | Input parameters: (string, string[, int])   (string IN string)              |
|                   | Output parameters: int                                                          | Output parameters: int                                                      |
| POSITIVE          | Input parameters: bigint double decimal                                         | Input parameters: bigint double decimal                                     |
|                   | Output parameters: Same as input parameters                                     | Output parameters: Same as input parameters                                 |
| POSEXPLODE        | Input parameters: array                                                         | –                                                                           |
|                   | Output parameters: table                                                        |                                                                             |
| POSEXPLODE_OUTER  | Input parameters: array                                                         | –                                                                           |
|                   | Output parameters: table                                                        |                                                                             |
| POW               | Input parameters: (bigint double decimal, bigint double decimal)                | Input parameters: (bigint double decimal, bigint double decimal)            |
|                   | Output parameters: double                                                       | Output parameters: double                                                   |
| POWER             | Input parameters: (bigint double decimal, bigint double decimal)                | Input parameters: (bigint double decimal, bigint double decimal)            |
|                   | Output parameters: double                                                       | Output parameters: double                                                   |

|                    |                                                   |                                            |
|--------------------|---------------------------------------------------|--------------------------------------------|
| PRINTF             | Input parameters: (string, T, ...)                | Input parameters: (string, T, ...)         |
|                    | Output parameters: string                         | Output parameters: string                  |
| QUARTER            | Input parameters: string date timestamp           | Input parameters: string date timestamp    |
|                    | Output parameters: int                            | Output parameters: int                     |
| RADIANS            | Input parameters: bigint double decimal           | Input parameters: bigint double decimal    |
|                    | Output parameters: double                         | Output parameters: double                  |
| RAISE_ERROR        | Input parameters: string                          | Input parameters: string                   |
|                    | Output parameters: throws an exception            | Output parameters: throws an exception     |
| RAND               | Input parameters: [int]                           | Input parameters: [int]                    |
|                    | Output parameters: double                         | Output parameters: double                  |
| RANDN              | Input parameters: [int]                           | Input parameters: [long]                   |
|                    | Output parameters: double                         | Output parameters: double                  |
| RANDOM             | Input parameters: [int]                           | Input parameters: [int]                    |
|                    | Output parameters: double                         | Output parameters: double                  |
| RANK               | Input parameters: None                            | Input parameters: None                     |
|                    | Output parameters: int                            | Output parameters: int                     |
| REGEXP             | Input parameters: (string, string)                | Input parameters: (string, string)         |
|                    | Output parameters: boolean                        | Output parameters: boolean                 |
| REGEXP_COUNT       | Input parameters: (string, string)                | –                                          |
|                    | Output parameters: int                            |                                            |
| REGEXP_EXTRACT     | Input parameters: (string, string[, int])         | Input parameters: (string, string, int)    |
|                    | Output parameters: int                            | Output parameters: int                     |
| REGEXP_EXTRACT_ALL | Input parameters: (string, string[, int])         | Input parameters: (string, string[, int])  |
|                    | Output parameters: array                          | Output parameters: array                   |
| REGEXP_LIKE        | Input parameters: (string, string)                | Input parameters: (string, string)         |
|                    | Output parameters: boolean                        | Output parameters: boolean                 |
| REGEXP_REPLACE     | Input parameters: (string, string, string[, int]) | Input parameters: (string, string, string) |
|                    | Output parameters: string                         | Output parameters: string                  |
| REGEXP_SUBSTR      | Input parameters: (string, string)                | –                                          |
|                    | Output parameters: string                         |                                            |
| REPEAT             | Input parameters: (string, int)                   | Input parameters: (string, int)            |
|                    | Output parameters: string                         | Output parameters: string                  |
| REPLACE            | Input parameters: (string, string[, string])      | Input parameters: (string, string, string) |
|                    | Output parameters: string                         | Output parameters: string                  |
| REVERSE            | Input parameters: string array                    | Input parameters: string                   |
|                    | Output parameters: string                         | Output parameters: string                  |
| RIGHT              | Input parameters: (string, int)                   | Input parameters: (string, int)            |
|                    | Output parameters: string                         | Output parameters: string                  |
| RINT               | Input parameters: bigint double decimal           | Input parameters: bigint double decimal    |
|                    | Output parameters: double                         | Output parameters: double                  |
| RLIKE              | Input parameters: (string, string)                | Input parameters: (string, string)         |
|                    | Output parameters: boolean                        | Output parameters: boolean                 |
| ROUND              | Input parameters: (double, int)                   | Input parameters: (double, int)            |
|                    | Output parameters: double                         | Output parameters: double                  |
| ROW_NUMBER         | Input parameters: None                            | Input parameters: None                     |
|                    | Output parameters: int                            | Output parameters: int                     |



|                    | Input parameters: (string, int[, string])                                     | Output parameters: string                                                     |
|--------------------|-------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| RPAD               | Input parameters: (string, int[, string])                                     | Output parameters: string                                                     |
|                    | Output parameters: string                                                     | Output parameters: string                                                     |
| RTRIM              | Input parameters: string                                                      | Input parameters: string                                                      |
|                    | Output parameters: string                                                     | Output parameters: string                                                     |
| SAMPLE             | Input parameters: (long [, long [, cols...]])                                 | –                                                                             |
|                    | Output parameters: boolean                                                    |                                                                               |
| SCHEMA_OF_CSV      | Input parameters: (string[, map])                                             |                                                                               |
|                    | Output parameters: struct                                                     |                                                                               |
| SCHEMA_OF_JSON     | Input parameters: (string[, map])                                             | –                                                                             |
|                    | Output parameters:                                                            |                                                                               |
| SECOND             | Input parameters: string timestamp                                            | Input parameters: string timestamp                                            |
|                    | Output parameters: int                                                        | Output parameters: int                                                        |
| SENTENCES          | (string[, string, string])                                                    | Input parameters: (string[, string, string])                                  |
|                    | Output parameters: array<array<string>>                                       | Output parameters: array<array<string>>                                       |
| SEQUENCE           | Input parameters: (int bigint date timestamp, int bigint date timestamp, int) | Input parameters: (int bigint date timestamp, int bigint date timestamp, int) |
|                    | Output parameters: Same as input parameters                                   | Output parameters: Same as input parameters                                   |
| SHA                | Input parameters: string binary                                               | Input parameters: string binary                                               |
|                    | Output parameters: string                                                     | Output parameters: string                                                     |
| SHA1               | Input parameters: string binary                                               | Input parameters: string binary                                               |
|                    | Output parameters: string                                                     | Output parameters: string                                                     |
| SHA2               | Input parameters: (string, int)                                               | Input parameters: (string, int)                                               |
|                    | Output parameters: string                                                     | Output parameters: string                                                     |
| SHIFLEFT           | Input parameters: (int bigint, int)                                           | Input parameters: (int biging, int)                                           |
|                    | Output parameters: int bigint                                                 | Output parameters: int bigint                                                 |
| SHIFTRIGHT         | Input parameters: (int bigint, int)                                           | Input parameters: (int bigint, int)                                           |
|                    | Output parameters: int bigint                                                 | Output parameters: int bigint                                                 |
| SHIFTRIGHTUNSIGNED | Input parameters: (int bigint, int)                                           | Input parameters: (int bigint, int)                                           |
|                    | Output parameters: int bigint                                                 | Output parameters: int bigint                                                 |
| SHUFFLE            | Input parameters: array                                                       | Input parameters: array                                                       |
|                    | Output parameters: array                                                      | Output parameters: array                                                      |
| SIGN               | Input parameters: bigint double decimal                                       | Input parameters: bigint double decimal                                       |
|                    | Output parameters: double                                                     | Output parameters: double                                                     |
| SIGNUM             | Input parameters: bigint double decimal                                       | Input parameters: bigint double decimal                                       |
|                    | Output parameters: double                                                     | Output parameters: double                                                     |
| SIN                | Input parameters: bigint double decimal                                       | Input parameters: bigint double decimal                                       |
|                    | Output parameters: double                                                     | Output parameters: double                                                     |
| SINH               | Input parameters: bigint double decimal                                       | Input parameters: bigint double decimal                                       |
|                    | Output parameters: double                                                     | Output parameters: double                                                     |
| SIZE               | Input parameters: array   map                                                 | Input parameters: array   map                                                 |
|                    | Output parameters: int                                                        | Output parameters: int                                                        |
| SKEWNESS           | Input parameters: bigint double decimal                                       | –                                                                             |
|                    | Output parameters: double                                                     |                                                                               |
| SLICE              | Input parameters: (array, int, int)                                           | Input parameters: (array, int, int)                                           |
|                    | Output parameters: array                                                      | Output parameters: array                                                      |
| SPLIT              | Input parameters: (string, int[, string])                                     | Input parameters: (string, int[, string])                                     |
|                    | Output parameters: array                                                      | Output parameters: array                                                      |



| SMALLINT           | cast as smallint                                                     | cast as smallint                                                                   |
|--------------------|----------------------------------------------------------------------|------------------------------------------------------------------------------------|
| SOME               | Input parameters: boolean                                            | Input parameters: boolean                                                          |
|                    | Output parameters: boolean                                           | Output parameters: boolean                                                         |
| SORT_ARRAY         | Input parameters: (array[, boolean])                                 | Input parameters: array                                                            |
|                    | Output parameters: array                                             | Output parameters: array                                                           |
| SOUNDEX            | Input parameters: string                                             | Input parameters: string                                                           |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SPACE              | Input parameters: string                                             | Input parameters: string                                                           |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SPARK_PARTITION_ID | Input parameters: None                                               | -                                                                                  |
|                    | Output parameters: int                                               |                                                                                    |
| SPLIT              | Input parameters: (string, string, int)                              | Input parameters: (string, string)                                                 |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SQRT               | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| STACK              | Input parameters: (T, U, ...)                                        | -                                                                                  |
|                    | Output parameters: table                                             |                                                                                    |
| STD                | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| STDDEV             | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| STDDEV_POP         | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| STDDEV_SAMP        | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| STRING             | cast as string                                                       | cast as string                                                                     |
| STRUCT             | Input parameters: (T, U, ...)                                        | Input parameters: (T, U, ...)                                                      |
|                    | Output parameters: struct                                            | Output parameters: struct                                                          |
| STR_TO_MAP         | Input parameters: (string[, string[, string]])                       | Input parameters: (string[, string, string])                                       |
|                    | Output parameters: map                                               | Output parameters: map                                                             |
| SUBSTR             | Input parameters: (string, int[, int])   (string FROM int [FOR int]) | Input parameters: (string binary, int[, int])   (string binary FROM int [FOR int]) |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SUBSTRING          | Input parameters: (string, int[, int])   (string FROM int [FOR int]) | Input parameters: (string binary, int[, int])   (string binary FROM int [FOR int]) |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SUBSTRING_INDEX    | Input parameters: (string, string, int)                              | Input parameters: (string, string, int)                                            |
|                    | Output parameters: string                                            | Output parameters: string                                                          |
| SUM                | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: Same as the input parameter                       | Output parameters: double                                                          |
| TAN                | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| TANH               | Input parameters: bigint double decimal                              | Input parameters: bigint double decimal                                            |
|                    | Output parameters: double                                            | Output parameters: double                                                          |
| TIMESTAMP          | Input parameters: string                                             | Input parameters: string                                                           |
|                    | Output parameters: timestamp                                         | Output parameters: timestamp                                                       |
| TIMESTAMP_ADD      | Input parameters: (date/timestamp, int, string)                      |                                                                                    |

|                   |                                                                                                |                                                                         |
|-------------------|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
|                   | Output parameters: timestamp                                                                   |                                                                         |
| TIMESTAMP_MICROS  | Input parameters: bigint                                                                       | Input parameters: bigint                                                |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TIMESTAMP_MILLIS  | Input parameters: bigint                                                                       | Input parameters: bigint                                                |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TIMESTAMP_SECONDS | Input parameters: bigint                                                                       | Input parameters: bigint                                                |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TIMEZONE_HOUR     | Input parameters: string                                                                       | -                                                                       |
|                   | Output parameters: int                                                                         |                                                                         |
| TIMEZONE_MINUTE   | Input parameters: string                                                                       | -                                                                       |
|                   | Output parameters: int                                                                         |                                                                         |
| TINYINT           | cast as tinyint                                                                                | cast as tinyint                                                         |
| TO_CHAR           | Input parameters: (boolean int long double decimal [, string])                                 |                                                                         |
|                   | Output parameters: string                                                                      |                                                                         |
| TO_CSV            | Input parameters: (struct[, map])                                                              | -                                                                       |
|                   | Output parameters: string                                                                      |                                                                         |
| TO_DATE           | Input parameters: (string[, string])                                                           | Input parameters: string                                                |
|                   | Output parameters: date                                                                        | Output parameters: date                                                 |
| TO_JSON           | Input parameters: (string[, map])                                                              | -                                                                       |
|                   | Output parameters: string                                                                      |                                                                         |
| TO_TIMESTAMP      | Input parameters: (string[, string])                                                           | Input parameters: (string)                                              |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TO_UNIX_TIMESTAMP | Input parameters: (string[, string])                                                           | Input parameters: (string[, string])                                    |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TO_UTC_TIMESTAMP  | Input parameters: (string[, string])                                                           | Input parameters: (bigint double decimal date timestamp string, string) |
|                   | Output parameters: timestamp                                                                   | Output parameters: timestamp                                            |
| TRANS_ARRAY       | Input parameters: (int, string, cols...)                                                       | -                                                                       |
|                   | Output parameters: Same type as cols                                                           |                                                                         |
| TRANS_COLS        | Input parameters: (int, cols...)                                                               |                                                                         |
|                   | Output parameters: (int, cols)                                                                 |                                                                         |
| TRANSLATE         | Input parameters: (string, string, string)                                                     | -                                                                       |
|                   | Output parameters: string                                                                      |                                                                         |
| TRIM              | Input parameters: string                                                                       | Input parameters: string                                                |
|                   | Output parameters: string                                                                      | Output parameters: string                                               |
| TRUNC             | Input parameters: (string date timestamp, string)                                              | Input parameters: (string date timestamp, string)                       |
|                   | Output parameters: string                                                                      | Output parameters: string                                               |
| TRY_ADD           | Input parameters: (bigint double decimal date timestamp, bigint double decimal date timestamp) | Input parameters: (bigint double decimal, bigint double decimal)        |
|                   | Output parameters: Same as input parameters                                                    | Output parameters: Same as input parameters                             |
| TRY_DIVIDE        | Input parameters: (bigint double decimal, bigint double decimal)                               | -                                                                       |
|                   | Output parameters: double                                                                      |                                                                         |
| TYPEOF            | Input parameters: T                                                                            | Input parameters: T                                                     |
|                   | Output parameters: string                                                                      | Output parameters: string                                               |
| UCASE             | Input parameters: string                                                                       | Input parameters: string                                                |
|                   | Output parameters: string                                                                      | Output parameters: string                                               |

|                |                                                                                              |                                                                                              |
|----------------|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| UNBASE64       | Input parameters: string                                                                     | Input parameters: string                                                                     |
|                | Output parameters: string                                                                    | Output parameters: string                                                                    |
| UNHEX          | Input parameters: string                                                                     | Input parameters: string                                                                     |
|                | Output parameters: binary                                                                    | Output parameters: binary                                                                    |
| UNIX_DATE      | Input parameters: string timestamp date                                                      | Input parameters: string timestamp date                                                      |
|                | Output parameters: int                                                                       | Output parameters: int                                                                       |
| UNIX_MICROS    | Input parameters: timestamp                                                                  | Input parameters: timestamp                                                                  |
|                | Output parameters: bigint                                                                    | Output parameters: bigint                                                                    |
| UNIX_MILLIS    | Input parameters: timestamp                                                                  | Input parameters: timestamp                                                                  |
|                | Output parameters: bigint                                                                    | Output parameters: bigint                                                                    |
| UNIX_SECONDS   | Input parameters: timestamp                                                                  | Input parameters: timestamp                                                                  |
|                | Output parameters: bigint                                                                    | Output parameters: bigint                                                                    |
| UNIX_TIMESTAMP | Input parameters: [date timestamp string[, string]]                                          | Input parameters: (date timestamp string[, string])                                          |
|                | Output parameters: bigint                                                                    | Output parameters: bigint                                                                    |
| UPPER          | Input parameters: string                                                                     | Input parameters: string                                                                     |
|                | Output parameters: string                                                                    | Output parameters: string                                                                    |
| URL_DECODE     | Input parameters: string                                                                     | –                                                                                            |
|                | Output parameters: string                                                                    |                                                                                              |
| UUID           | Input parameters: None                                                                       | Input parameters: None                                                                       |
|                | Output parameters: string                                                                    | Output parameters: string                                                                    |
| VARIANCE       | Input parameters: bigint double decimal                                                      | Input parameters: bigint double decimal                                                      |
|                | Output parameters: double                                                                    | Output parameters: double                                                                    |
| VAR_POP        | Input parameters: bigint double decimal                                                      | Input parameters: bigint double decimal                                                      |
|                | Output parameters: double                                                                    | Output parameters: double                                                                    |
| VAR_SAMP       | Input parameters: bigint double decimal                                                      | Input parameters: bigint double decimal                                                      |
|                | Output parameters: double                                                                    | Output parameters: double                                                                    |
| VERSION        | Input parameters: None                                                                       | Input parameters: None                                                                       |
|                | Output parameters: string                                                                    | Output parameters: string                                                                    |
| WEEKDAY        | Input parameters: date timestamp string                                                      | Input parameters: date timestamp string                                                      |
|                | Output parameters: int                                                                       | Output parameters: int                                                                       |
| WEEKOFYEAR     | Input parameters: date timestamp string                                                      | Input parameters: date timestamp string                                                      |
|                | Output parameters: int                                                                       | Output parameters: int                                                                       |
| WIDTH_BUCKET   | Input parameters: (bigint double decimal, bigint double decimal, bigint double decimal, int) | Input parameters: (bigint double decimal, bigint double decimal, bigint double decimal, int) |
|                | Output parameters: int                                                                       | Output parameters: int                                                                       |
| WINDOW         | Input parameters: (date timestamp, string[, string[, string]])                               | –                                                                                            |
|                | Output parameters: table                                                                     |                                                                                              |
| XPath          | Input parameters: (string, string)                                                           | Input parameters: (string, string)                                                           |
|                | Output parameters: array                                                                     | Output parameters: array                                                                     |
| XPath_BOOLEAN  | Input parameters: (string, string)                                                           | Input parameters: (string, string)                                                           |
|                | Output parameters: boolean                                                                   | Output parameters: boolean                                                                   |
| XPath_DOUBLE   | Input parameters: (string, string)                                                           | Input parameters: (string, string)                                                           |
|                | Output parameters: double                                                                    | Output parameters: double                                                                    |
| XPath_FLOAT    | Input parameters: (string, string)                                                           | Input parameters: (string, string)                                                           |
|                | Output parameters: float                                                                     | Output parameters: float                                                                     |
| XPath_INT      | Input parameters: (string, string)                                                           | Input parameters: (string, string)                                                           |

|              |                                         |                                         |
|--------------|-----------------------------------------|-----------------------------------------|
| XPATH_INT    | Output parameters: int                  | Output parameters: int                  |
|              |                                         |                                         |
| XPATH_LONG   | Input parameters: (string, string)      | Input parameters: (string, string)      |
|              | Output parameters: bigint               | Output parameters: bigint               |
| XPATH_NUMBER | Input parameters: (string, string)      | Input parameters: (string, string)      |
|              | Output parameters: double               | Output parameters: double               |
| XPATH_STRING | Input parameters: (string, string)      | Input parameters: (string, string)      |
|              | Output parameters: string               | Output parameters: string               |
| XXHASH64     | Input parameters: (T, U, ...)           | Input parameters: (T, U, ...)           |
|              | Output parameters: bigint               | Output parameters: bigint               |
| YEAR         | Input parameters: date timestamp string | Input parameters: date timestamp string |
|              | Output parameters: int                  | Output parameters: int                  |

# Binary Functions

Last updated: 2026-05-21 14:30:50

## CRC32

- Function Syntax:

```
CRC32(<expr> string|binary)
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Calculate the cyclic redundancy check value of an expression using the CRC32 algorithm.
- Return type: bigint
- Example:

```
> select crc32('tencent');  
820633257
```

## MD5

- Function Syntax:

```
MD5(<expr string|binary>)
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Return the MD5 128-bit checksum as a hexadecimal string.
- Return type: string
- Example:

```
> select md5('tencent');  
3da576879001c77b442b9f8ef95c09d6
```

## HASH

- Function Syntax:

```
HASH(<expr1> any[, <expr2> any, ...])
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Returns the hash value of all parameters. SPARKSQL and PRESTO use different calculation methods, which may yield different results.
- Return type: integer
- Example:

```
> SELECT hash('tencent', array(123), 2);  
-412995102
```

## XXHASH64

- Function Syntax:

```
XXHASH64(<expr1> any[, <expr2> any, ...])
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Returns the 64-bit hash value of the parameters. SPARKSQL and PRESTO use different calculation methods, which may yield different results.
- Return type: bigint
- Example:

```
> SELECT xxhash64('tencent', array(123), 2);  
-1900074178543885261
```

## SHA

- Function Syntax:

```
SHA(<expr> string|binary)
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Returns the sha1 hash value of `expr` as a hexadecimal string.
- Return type: string
- Example:

```
> select sha('tencent');  
f94b2c96e2f127726ef4bcec6bc779f0f2e7888f
```

## SHA1

- Function Syntax:

```
SHA1(<expr> string|binary)
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Returns the sha1 hash value of `expr` as a hexadecimal string.
- Return type: `string`
- Example:

```
> select sha1('tencent');  
f94b2c96e2f127726ef4bcec6bc779f0f2e7888f
```

## SHA2

- Function Syntax:

```
SHA2(expr string|binary, bitLength int)
```

- Supported engines: SPARKSQL, PRESTO.
- Usage: Returns the checksum of the SHA-2 family for `expr` as a hexadecimal string. It supports SHA-224, SHA-256, SHA-384, and SHA-512. A bit length of 0 is equivalent to 256.
- Return type: `string`
- Example:

```
> select sha2('tencent', 256);  
9c8ae69b84f21a2e46df9edf0063a697afec050188ff2884ddc8ab32b5e58c43
```

## AES\_ENCRYPT

- Function Syntax:

```
AES_ENCRYPT(<expr> string|binary, <key> string|binary)
```

- Supported engines: PRESTO.
- Usage: Encrypts `expr` using the AES algorithm.
- Return type: `binary`
- Example:

```
> SELECT hex(aes_encrypt('tencent', '0000111122223333'));  
B99B99CE3359A736DBB9811ED8815C01
```

## AES\_DECRYPT

- Function Syntax:

```
AES_DECRYPT(<expr> string|binary, <key> string|binary)
```

- Supported engines: PRESTO
- Usage: Decrypts `expr` using the AES algorithm.
- Return type: `binary`
- Example:

```
> SELECT aes_decrypt(unhex('B99B99CE3359A736DBB9811ED8815C01'),  
'0000111122223333');  
tencent
```



# Bitwise Operation Functions

Last updated: 2026-05-21 14:31:07

## BIT\_COUNT

- Function Syntax:

```
BIT_COUNT ( <expr> bigint | boolean )
```

- Supported engines: SparkSQL, Presto.
- Usage: Treat `expr` as an unsigned 64-bit integer and return the number of bits set to 1. If the argument is NULL, NULL is returned.
- Return type: `integer`
- Example:

```
> SELECT bit_count(5);  
2
```

## BIT\_GET

- Function Syntax:

```
BIT_GET ( <expr> bigint | boolean, <pos> integer )
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the value of the bit at the specified position (0 or 1). Positions are numbered from right to left, starting from zero. The position argument cannot be negative.
- Return type: `integer`
- Example:

```
> SELECT bit_get(11, 0);  
1  
> SELECT bit_get(11, 2);  
0
```

## GETBIT

- **Function Syntax:**

```
GETBIT ( <expr> bigint | boolean, <pos> integer )
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the value of the bit at the specified position (0 or 1). Positions are numbered from right to left, starting from zero. The position argument cannot be negative.
- **Return type:** `integer`
- **Example:**

```
> SELECT getbit(11, 0);  
1  
> SELECT getbit(11, 2);  
0
```

# Aggregate Functions

Last updated: 2026-05-21 14:31:27

## ARRAY

- Function Syntax:

```
ARRAY(<e1> T, ..., <en> T)
```

- Supported engines: SparkSQL, Presto.
- Usage: Generates an array from the given elements.
- Return type: array<T>.
- Example:

```
> SELECT array(1, 2, 3);  
[1,2,3]
```

## FILTER

- Function Syntax:

```
FILTER(<expr> array<T>, <predicate> function(T[, integer])->boolean)
```

- Supported engines: SparkSQL.
- Usage: Filters the input array using the given predicate.
- Return type: array<T>.
- Example:

```
> SELECT `filter`(array(1, 2, 3), x -> x % 2 == 1);  
[1,3]  
> SELECT `filter`(array(0, 2, 3), (x, i) -> x > i);  
[2,3]  
> SELECT `filter`(array(0, null, 2, 3, null), x -> x IS NOT NULL);  
[0,2,3]
```

## TRANSFORM

- **Function Syntax:**

```
TRANSFORM(<expr> array<T>, <func> function(T[, integer])->U)
```

- **Supported engines:** SparkSQL.
- **Usage:** Transforms elements in the array using the func.
- **Return type:** array<U>.
- **Example:**

```
> SELECT transform(array(1, 2, 3), x -> x + 1);
[2,3,4]
> SELECT transform(array(1, 2, 3), (x, i) -> x + i);
[1,3,5]
```

## ZIP\_WITH

- **Function Syntax:**

```
ZIP_WITH(<left> array<T>, <right> array<U>, <func> function(T, U)->R)
```

- **Supported engines:** SparkSQL.
- **Usage:** Merges two given arrays element-wise into a single array using a function. If one array is shorter, nulls are appended to its end before the function is applied to match the length of the longer array.
- **Return type:** array<R>.
- **Example:**

```
> SELECT zip_with(array(1, 2, 3), array('a', 'b', 'c'), (x, y) -> (y,
x));
[{"y":"a","x":1},{ "y":"b","x":2},{ "y":"c","x":3}]
> SELECT zip_with(array(1, 2), array(3, 4), (x, y) -> x + y);
[4,6]
> SELECT zip_with(array('a', 'b', 'c'), array('d', 'e', 'f'), (x, y) -
> concat(x, y));
["ad","be","cf"]
```

## FORALL

- **Function Syntax:**

```
FORALL(<expr> array<T>, <pred> function(T)->boolean)
```

- **Supported engines:** SparkSQL.
- **Usage:** Tests whether the predicate applies to all elements in the array.
- **Return type:** boolean.
- **Example:**

```
> SELECT forall(array(1, 2, 3), x -> x % 2 == 0);  
false  
> SELECT forall(array(2, 4, 8), x -> x % 2 == 0);  
true  
> SELECT forall(array(1, null, 3), x -> x % 2 == 0);  
false  
> SELECT forall(array(2, null, 8), x -> x % 2 == 0);  
NULL
```

## AGGREGATE

- **Function Syntax:**

```
AGGREGATE(<expr> array<T>, <start> U, <merge> function(U, T)->U,  
<finish> function(U)->R)
```

- **Supported engines:** SparkSQL.
- **Usage:** Aggregates the elements in the array expr.
- **Return type:** R.
- **Example:**

```
> SELECT aggregate(array(1, 2, 3), 0, (acc, x) -> acc + x);  
6  
> SELECT aggregate(array(1, 2, 3), 0, (acc, x) -> acc + x, acc -> acc  
* 10);  
60
```

## EXISTS

- **Function Syntax:**

```
EXISTS(<expr> array<T>, <pred> function(T)->boolean)
```

- **Supported engines:** SparkSQL.
- **Usage:** Tests whether the predicate applies to one or more elements in the array.
- **Return type:** boolean.
- **Example:**

```
> SELECT exists(array(1, 2, 3), x -> x % 2 == 0);
true> SELECT exists(array(1, 2, 3), x -> x % 2 == 10);
false
> SELECT exists(array(1, null, 3), x -> x % 2 == 0);
NULL
> SELECT exists(array(0, null, 2, 3, null), x -> x IS NULL);
true
> SELECT exists(array(1, 2, 3), x -> x IS NULL);
false
```

## ARRAY\_CONTAINS

- **Function Syntax:**

```
ARRAY_CONTAINS(<expr> array<T>, <value> T)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns true if the array contains the value.
- **Return type:** boolean.
- **Example:**

```
> SELECT array_contains(array(1, 2, 3), 2);
true
```

## ARRAYS\_OVERLAP

- **Function Syntax:**

```
ARRAYS_OVERLAP(<a> array<T>, <b> array<U>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if array a contains at least one non-null element that also exists in array b. Returns null if the arrays have no common elements, both are non-empty, and either contains a null element. Otherwise, returns false.
- Return type: boolean.
- Example:

```
> SELECT arrays_overlap(array(1, 2, 3), array(3, 4, 5));  
true
```

## ARRAY\_INTERSECT

- Function Syntax:

```
ARRAY_INTERSECT(<a> array<T>, <b> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns an array of elements at the intersection of a and b, with no duplicates.
- Return type: array<T>.
- Example:

```
> SELECT array_intersect(array(1, 2, 3), array(1, 3, 5));  
[1, 3]
```

## ARRAY\_JOIN

- Function Syntax:

```
ARRAY_JOIN(<a> ARRAY<T>, <delimiter> string[, <nullReplacement>  
string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Joins the elements of a given array using a delimiter and an optional string to replace nulls. If no value is set for nullReplacement, all null values are filtered out.
- Return type: string.

- Example:

```
> SELECT array_join(array('hello', 'world'), ' ');
hello world
> SELECT array_join(array('hello', null , 'world'), ' ');
hello world
> SELECT array_join(array('hello', null , 'world'), ' ', ',');
hello , world
```

## ARRAY\_POSITION

- Function Syntax:

```
ARRAY_POSITION(<a> array<T>, <element> T)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the index (starting from 1) of the first element in the array.
- Return type: integer.
- Example:

```
> SELECT array_position(array(3, 2, 1), 1);
3
```

## ARRAY\_SORT

- Function Syntax:

```
ARRAY_SORT(<a> array<T>[, <func> function(T, T)->integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Sorts the input array. If func is omitted, sorts in ascending order.
- Return type: array<T>.
- Example:

```
> SELECT array_sort(array(5, 6, 1), (left, right) -> case when left <
right then -1 when left > right then 1 else 0 end);
[1,5,6]
```



```
> SELECT array_sort(array('bc', 'ab', 'dc'), (left, right) -> case
when left is null and right is null then 0 when left is null then -1
when right is null then 1 when left < right then 1 when left > right
then -1 else 0 end);
["dc","bc","ab"]
> SELECT array_sort(array('b', 'd', null, 'c', 'a'));
["a","b","c","d",null]
```

## ARRAY\_EXCEPT

- Function Syntax:

```
ARRAY_EXCEPT(<a> array<T>, <b> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns an array of elements that are in a but not in b, with no duplicates.
- Return type: array<T>.
- Example:

```
> SELECT array_except(array(1, 2, 3), array(1, 3, 5));
[2]
```

## ARRAY\_UNION

- Function Syntax:

```
ARRAY_UNION(<a> array<T>, <b> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns an array of elements that are in the union of a and b, with no duplicates.
- Return type: array<T>.
- Example:

```
> SELECT array_union(array(1, 2, 3), array(1, 3, 5));
[1,2,3,5]
```

## NAMED\_STRUCT

- **Function Syntax:**

```
NAMED_STRUCT(name1 K, val1 V, ...)
```

- **Supported engines:** SparkSQL.
- **Usage:** Creates a struct using the given field names and values.
- **Return type:** struct.
- **Example:**

```
> SELECT named_struct("a", 1, "b", 2, "c", 3);  
{ "a":1, "b":2, "c":3 }
```

## STRUCT

- **Function Syntax:**

```
STRUCT(<col1> T1, <col2> T2, ...)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Creates a struct using the given field names and values.
- **Return type:** struct.
- **Example:**

```
> SELECT struct('a', 'b', 'c');  
{ "col1":"a", "col2":"b", "col3":"c" }  
  
> SELECT struct('a', 'b', 'c', 1, 2);  
{ "col1":"a", "col2":"b", "col3":"c", "col4":1, "col5":2 }
```

## SLICE

- **Function Syntax:**

```
SLICE(<a> array<T>, <start> integer, <length> integer)
```

- **Supported engines:** SparkSQL, Presto.

- **Usage:** Returns a subset of array `a` starting from index `start` (array indices start from 1; if `start` is negative, it counts from the end) with the specified length `length`.
- **Return type:** array<T>.
- **Example:**

```
> SELECT slice(array(1, 2, 3, 4), 2, 2);  
[2, 3]  
> SELECT slice(array(1, 2, 3, 4), -2, 2);  
[3, 4]
```

## ARRAYS\_ZIP

- **Function Syntax:**

```
ARRAYS_ZIP(<a1> array<T>, ...)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns a merged array whose elements are of type struct, where the Nth struct contains the Nth value from each input array.
- **Return type:** array<struct<string, T>>.
- **Example:**

```
> SELECT arrays_zip(array(1, 2, 3), array(2, 3, 4));  
[{"0":1,"1":2},{ "0":2,"1":3},{ "0":3,"1":4}]  
> SELECT arrays_zip(array(1, 2), array(2, 3), array(3, 4));  
[{"0":1,"1":2,"2":3},{ "0":2,"1":3,"2":4}]
```

## SORT\_ARRAY

- **Function Syntax:**

```
SORT_ARRAY(<a> array<T>[, ascendingOrder boolean])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Sorts the input array in ascending or descending order. For the double/float type, NaN is greater than any non-NaN element. Null elements are placed at the beginning of the returned array for ascending order, or at the end for descending order.
- **Return type:** array<T>.

- Example:

```
> SELECT sort_array(array('b', 'd', null, 'c', 'a'), true);  
[null,"a","b","c","d"]
```

## SHUFFLE

- Function Syntax:

```
SHUFFLE(<a> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a random permutation of the given array.
- Return type: array<T>.
- Example:

```
> SELECT shuffle(array(1, 20, 3, 5));  
[3,1,5,20]  
> SELECT shuffle(array(1, 20, null, 3));  
[20,null,3,1]
```

## ARRAY\_MAX

- Function Syntax:

```
ARRAY_MAX(<a> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the maximum value in the array. For the double/float type, NaN is greater than any non-NaN element. Null elements are skipped.
- Return type: array<T>.
- Example:

```
> SELECT array_max(array(1, 20, null, 3));  
20
```

## ARRAY\_MIN

- **Function Syntax:**

```
ARRAY_MIN(<a> array<T>)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the minimum value in the array. For the double/float type, NaN is greater than any non-NaN element. Null elements are skipped.
- **Return type:** array<T>.
- **Example:**

```
> SELECT array_min(array(1, 20, null, 3));  
1
```

## FLATTEN

- **Function Syntax:**

```
FLATTEN(<aa> array<array<T>>)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Converts a two-dimensional array into a one-dimensional array.
- **Return type:** array<T>.
- **Example:**

```
> SELECT flatten(array(array(1, 2), array(3, 4)));  
[1,2,3,4]
```

## SEQUENCE

- **Function Syntax:**

```
SEQUENCE(<start> integer|date|timestamp, end integer|date|timestamp[,  
step integer|interval])
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Generates an array from `start` to `end` (inclusive), incrementing stepwise. The type of the returned elements is the same as the type of `start` and `end`.

The `start` and `stop` expressions must be resolved to the same type. If the start and stop expressions are resolved to the date or timestamp type, the `step` must be resolved to the interval, year-month interval, or day-time interval type; otherwise, it is resolved to the same type as `start` and `end`.

- **Return type:** Same as start
- **Example:**

```
> SELECT sequence(1, 5);
[1,2,3,4,5]
> SELECT sequence(5, 1);
[5,4,3,2,1]
> SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'),
interval 1 month);
[2018-01-01,2018-02-01,2018-03-01]
> SELECT sequence(to_date('2018-01-01'), to_date('2018-03-01'),
interval '0-1' year to month);
[2018-01-01,2018-02-01,2018-03-01]
```

## ARRAY\_REPEAT

- **Function Syntax:**

```
ARRAY_REPEAT(<element> T, <count> integer)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns an array containing `count` elements.
- **Return type:** array<T>.
- **Example:**

```
> SELECT array_repeat('123', 2);
["123","123"]
```

## ARRAY\_REMOVE

- **Function Syntax:**

```
ARRAY_REMOVE(<a> array<T>, <element> T)
```

- Supported engines: SparkSQL, Presto.
- Usage: Removes all elements equal to the specified element from the array.
- Return type: array<T>.
- Example:

```
> SELECT array_remove(array(1, 2, 3, null, 3), 3);  
[1,2,null]
```

## ARRAY\_DISTINCT

- Function Syntax:

```
ARRAY_DISTINCT(<a> array<T>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Removes duplicate values from the array.
- Return type: array<T>.
- Example:

```
> SELECT array_distinct(array(1, 2, 3, null, 3));  
[1,2,3,null]
```

## ELEMENT\_AT

- Function Syntax:

```
ELEMENT_AT(<a> array<T>, <index> integer)  
ELEMENT_AT(<m> map<K, V>, <key> K)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the array element at the specified index (starting from 1) or the value for the specified key.
- Return Type: T, V
- Example:

```
> SELECT element_at(array(1, 2, 3), 2);  
2
```

```
> SELECT element_at(map(1, 'a', 2, 'b'), 2);  
b
```

## MAP

- Function Syntax:

```
MAP(<k1> K, <v1> V, ...)
```

- Supported engines: SparkSQL, Presto.
- Usage: Generates a map from the given elements.
- Return type: MAP<K, V>.
- Example:

```
> SELECT map(1.0, '2', 3.0, '4');  
{1.0:"2",3.0:"4"}
```

## MAP\_FROM\_ARRAYS

- Function Syntax:

```
MAP_FROM_ARRAYS(<keys> array<K>, <values> array<V>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Creates a map from a given pair of key/value arrays. No element in the keys array should be null.
- Return type: map<K, V>.
- Example:

```
> SELECT map_from_arrays(array(1.0, 3.0), array('2', '4'));  
{1.0:"2",3.0:"4"}
```

## MAP\_KEYS

- Function Syntax:

```
MAP_KEYS(<m> map<K, V>)
```



- Supported engines: SparkSQL, Presto.
- Usage: Returns an unordered array containing the keys of a map.
- Return type: array<K>.
- Example:

```
> SELECT map_keys(map(1, 'a', 2, 'b'));  
[1, 2]
```

## MAP\_VALUES

- Function Syntax:

```
MAP_VALUES (<m> map<K, V>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns an unordered array containing the values of a map.
- Return type: array<V>.
- Example:

```
> SELECT map_values(map(1, 'a', 2, 'b'));  
["a", "b"]
```

## MAP\_ENTRIES

- Function Syntax:

```
MAP_ENTRIES (<m> map<K, V>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns an unordered array containing all entries from a given map.
- Return type: array<struct<K, V>>.
- Example:

```
> SELECT map_entries(map(1, 'a', 2, 'b'));  
[{"key":1,"value":"a"}, {"key":2,"value":"b"}]
```

## MAP\_FROM\_ENTRIES

- **Function Syntax:**

```
MAP_FROM_ENTRIES(<entries> array<struct<K, V>>)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a map created from the given array of entries.
- **Return type:** map<K, V>.
- **Example:**

```
> SELECT map_from_entries(array(struct(1, 'a'), struct(2, 'b')));  
{1:"a",2:"b"}
```

## MAP\_CONCAT

- **Function Syntax:**

```
MAP_CONCAT(map1 map<K, V>, ...)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the union of all given maps.
- **Return type:** map<K, V>.
- **Example:**

```
> SELECT map_concat(map(1, 'a', 2, 'b'), map(3, 'c'));  
{1:"a",2:"b",3:"c"}
```

## MAP\_FILTER

- **Function Syntax:**

```
MAP_FILTER(<m> map<K, V>, <func> function(K, V)->boolean)
```

- **Supported engines:** SparkSQL.
- **Usage:** Filters entries in m using the func.
- **Return type:** map<K, V>.
- **Example:**

```
> SELECT map_filter(map(1, 0, 2, 2, 3, -1), (k, v) -> k > v);
{1:0,3:-1}
```

## MAP\_ZIP\_WITH

- Function Syntax:

```
MAP_ZIP_WITH(<map1> map<K, V1>, <map2> map<K, V2>, <func> function(K,
V1, V2)->R)
```

- Supported engines: SparkSQL.
- Usage: Merges two given maps into a single map using the func. For keys that appear in only one map, their values are set to NULL. If the input maps contain duplicate keys, only the first entry for each duplicate key is passed to the func.
- Return type: MAP<K, R>.
- Example:

```
> SELECT map_zip_with(map(1, 'a', 2, 'b'), map(1, 'x', 2, 'y'), (k,
v1, v2) -> concat(v1, v2));
{1:"ax",2:"by"}
```

## TRANSFORM\_KEYS

- Function Syntax:

```
TRANSFORM_KEYS(<m> map<K, V>, <func> function(K, V)->R)
```

- Supported engines: SparkSQL.
- Usage: Transforms keys in the map using the func.
- Return type: map<R, V>.
- Example:

```
> SELECT transform_keys(map_from_arrays(array(1, 2, 3), array(1, 2,
3)), (k, v) -> k + 1);
{2:1,3:2,4:3}
> SELECT transform_keys(map_from_arrays(array(1, 2, 3), array(1, 2,
3)), (k, v) -> k + v);
```

```
{2:1,4:2,6:3}
```

## TRANSFORM\_VALUES

- Function Syntax:

```
TRANSFORM_VALUES(<m> map<K, V>, <func> function(K, V)->R)
```

- Supported engines: SparkSQL.
- Usage: Transforms values in the map using the func.
- Return type: map<K, R>.
- Example:

```
> SELECT transform_values(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) -> v + 1);
      {1:2,2:3,3:4}
> SELECT transform_values(map_from_arrays(array(1, 2, 3), array(1, 2, 3)), (k, v) -> k + v);
      {1:2,2:4,3:6}
```

## SIZE

- Function Syntax:

```
SIZE(<expr> array<T>|map<K, V>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the size of an array or a map.
- Return type: integer.
- Example:

```
> SELECT size(array('b', 'd', 'c', 'a'));
      4
> SELECT size(map('a', 1, 'b', 2));
      2
> SELECT size(NULL);
     -1
```

## CARDINALITY

- Function Syntax:

```
CARDINALITY(<expr> array<T>|map<K, V>)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the size of an array or a map.
- Return type: integer.
- Example:

```
> SELECT cardinality(array('b', 'd', 'c', 'a'));
4
> SELECT cardinality(map('a', 1, 'b', 2));
2
> SELECT cardinality(NULL);
-1
```

## ANY\_MATCH / FORALL

- Function Syntax:

```
ANY_MATCH(<expr> array<T>, x -> lambda(x))
FORALL(<expr> array<T>, x -> lambda(x))
```

- Supported engines: SparkSQL.
- Usage: Executes a lambda expression for each element in the array sequentially. The function returns true if at least one execution returns true; otherwise, it returns false.
  - If any element in the array is NULL, the function returns NULL.
- Return type: boolean
- Example:

```
> SELECT any_match(array(1, 2, 3), x -> x % 2 == 0);
false
> SELECT any_match(array(2, 4, 8), x -> x % 2 == 0);
true
> SELECT any_match(array(1, null, 3), x -> x % 2 == 0);
```

NULL

# Date and Time Functions

Last updated: 2026-05-21 14:31:49

## DATE

- Function Syntax:

```
DATE(<expr> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Cast to date.
- Return type: date.
- Example:

```
> select date('2022-02-02');  
2022-02-02
```

## TIMESTAMP

- Function Syntax:

```
TIMESTAMP(<expr> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Cast to timestamp.
- Return type: timestamp.
- Example:

```
> select timestamp('2022-02-02 11:11:11');  
2022-02-02 11:11:11
```

## ADD\_MONTHS

- Function Syntax:

```
ADD_MONTHS(<start_date> date|timestamp|string, <num> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the date that is num months after start\_date.
- Return type: date.
- Example:

```
> SELECT add_months('2016-08-31', 1);  
2016-09-30
```

## CURRENT\_DATE

- Function Syntax:

```
CURRENT_DATE
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the current date at the start of query computation.
- Return type: date.
- Example:

```
> SELECT CURRENT_DATE;  
2022-07-27
```

## CURRENT\_TIMESTAMP

- Function Syntax:

```
CURRENT_TIMESTAMP
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the current timestamp at the start of query computation.
- Return type: timestamp.
- Example:

```
> SELECT CURRENT_TIMESTAMP;  
2022-07-27 18:06:00.632
```

## CURRENT\_TIMEZONE



- **Function Syntax:**

```
CURRENT_TIMEZONE (
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the local time zone of the current session.
- **Return type:** string.
- **Example:**

```
> select CURRENT_TIMEZONE();  
Asia/Shanghai
```

## DATEDIFF

- **Function Syntax:**

```
DATEDIFF(<end> date|timestamp|string, <start> date|timestamp|string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the number of days from start to end.
- **Return type:** integer.
- **Example:**

```
> SELECT datediff('2009-07-31', '2009-07-30');  
1  
> SELECT datediff('2009-07-30', '2009-07-31');  
-1
```

## DATE\_ADD

- **Function Syntax:**

```
DATE_ADD(<start_dates> date|timestamp|string, <num> integer)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the date that is num days after start\_date.
- **Return type:** date.

- Example:

```
> SELECT date_add('2016-07-30', 1);  
2016-07-31
```

## DATE\_FORMAT

- Function Syntax:

```
DATE_FORMAT(<ts> date|timestamp|string, <format> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts a timestamp to a string value in the specified date format.
- Return type: string.
- Example:

```
> SELECT date_format('2016-04-08', 'y');  
2016
```

## DATE\_SUB

- Function Syntax:

```
DATE_SUB(<start_date> date|timestamp|string, <num> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the date that is num days before start\_date.
- Return type: date.
- Example:

```
> SELECT date_sub('2016-07-30', 1);  
2016-07-29
```

## DAY

- Function Syntax:

```
DAY(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the day of the month for a given date/timestamp d.
- Return type: integer.
- Example:

```
> SELECT day('2009-07-30');  
30
```

## DAYOFYEAR

- Function Syntax:

```
DAYOFYEAR(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the day of the year for a given date/timestamp d.
- Return type: integer.
- Example:

```
> SELECT dayofyear('2016-04-09');  
100
```

## DAYOFMONTH

- Function Syntax:

```
DAYOFMONTH(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the day of the month for a given date/timestamp d.
- Return type: integer.
- Example:

```
> SELECT dayofmonth('2009-07-30');
```

## EXTRACT

- Function Syntax:

```
EXTRACT(<field> string FROM <source> date|timestamp|interval)
```

- Supported engines: SparkSQL, Presto.
- Usage: Extracts the part specified by field from source. SparkSQL supports source types of date/timestamp/interval, while Presto supports source types of date/timestamp.
  - For a source of type date/timestamp, the field can take the following values:
    - (1) "YEAR", ("Y", "YEARS", "YR", "YRS") – Obtains the year field from a given date.
    - (2) "YEAROFWEEK" – Obtains the ISO 8601 week-numbering year to which a given date belongs. For example, January 2, 2005 falls in week 53 of 2004, so the result is 2004.
    - (3) "QUARTER", ("QTR") – Obtains the quarter field from a given date.
    - (4) "MONTH", ("MON", "MONS", "MONTHS") – Obtains the month field from a given date.
    - (5) "WEEK", ("W", "WEEKS") – The week number within an ISO 8601 week-numbering year.
    - (6) "DAY", ("D", "DAYS") – Obtains the day field from a given date (1 – 31).
    - (7) "DAYOFWEEK", ("DOW") – Obtains the day number within the week for a given date, that is, Monday through Sunday.
    - (8) "DAYOFWEEK\_ISO", ("DOW\_ISO") – Represents the day of the week based on the ISO 8601 standard for date and time.
    - (9) "DOY" – Obtains the day number within a year for a given date (1 – 365/366).
    - (10) "HOUR", ("H", "HOURS", "HR", "HRS") – Obtains the hour field from a given date (0 – 23).
    - (11) "MINUTE", ("M", "MIN", "MINS", "MINUTES") – Obtains the minute field from a given date (0 – 59).
    - (12) "SECOND", ("S", "SEC", "SECONDS", "SECS") – Obtains the second field from a given date, including the fractional part.
  - For a source of type interval, the field can take the following values:
    - (1) "YEAR", ("Y", "YEARS", "YR", "YRS") – Total number of months / 12
    - (2) "MONTH", ("MON", "MONS", "MONTHS") – Total number of months % 12
    - (3) "DAY", ("D", "DAYS") – Obtains the number of days in a given interval.
    - (4) "HOUR", ("H", "HOURS", "HR", "HRS") – Number of hours contained in microseconds.

(5) "MINUTE", ("M", "MIN", "MINS", "MINUTES") – Extracts the remaining minutes after hours are extracted from microseconds.

(6) "SECOND", ("S", "SEC", "SECONDS", "SECS") – Extracts the remaining seconds after hours and minutes are extracted from microseconds.

- **Return Type**

- **SparkSQL:** The return type varies depending on the field type.
- **Presto:** bigint

- **Example:**

```
> SELECT extract(YEAR FROM TIMESTAMP '2019-08-12 01:00:00.123456');
2019
> SELECT extract(week FROM timestamp'2019-08-12 01:00:00.123456');
33
> SELECT extract(doy FROM DATE'2019-08-12');
224
> SELECT extract(SECONDS FROM timestamp'2019-10-01 00:00:01.000001');
1.000001
> SELECT extract(days FROM interval 5 days 3 hours 7 minutes);
5
> SELECT extract(seconds FROM interval 5 hours 30 seconds 1
milliseconds 1 microseconds);
30.001001
> SELECT extract(MONTH FROM INTERVAL '2021-11' YEAR TO MONTH);
11
> SELECT extract(MINUTE FROM INTERVAL '123 23:55:59.002001' DAY TO
SECOND);
55
```

## FROM\_UNIXTIME

- **Function Syntax:**

```
FROM_UNIXTIME(<unix_time> bigint[, <fmt> string])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the date/time represented by `unix_time` in the format `fmt`. If `fmt` is omitted, 'yyyy-MM-dd HH:mm:ss' is used.
- **Return type:** string.

- Example:

```
> SELECT from_unixtime(0, 'yyyy-MM-dd HH:mm:ss');
1969-12-31 16:00:00
> SELECT from_unixtime(0);
1969-12-31 16:00:00
```

## FROM\_UTC\_TIMESTAMP

- Function Syntax:

```
FROM_UTC_TIMESTAMP(<ts> timestamp, <timezone> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Takes a utc timestamp and renders it as a timestamp in the specified time zone.
- Return type: timestamp.
- Example:

```
> SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
2016-08-31 09:00:00
```

## HOUR

- Function Syntax:

```
HOUR(<ts> string|timestamp)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the hour part of the specified timestamp ts.
- Return type: integer.
- Example:

```
> SELECT hour('2009-07-30 12:58:59');
12
```

## LAST\_DAY

- Function Syntax:

```
LAST_DAY(<d> date|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the last day of the month for the given date d.
- Return type: date.
- Example:

```
> SELECT last_day('2009-01-12');  
2009-01-31
```

## MINUTE

- Function Syntax:

```
MINUTE(<ts> timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the minute of the specified timestamp ts.
- Return type: integer.
- Example:

```
> SELECT minute('2009-07-30 12:58:59');  
58
```

## MONTH

- Function Syntax:

```
MONTH(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the month of the specified date d.
- Return type: integer.
- Example:

```
> SELECT month('2016-07-30');
```

7

## MONTHS\_BETWEEN

- Function Syntax:

```
MONTHS_BETWEEN(<ts1> date|timestamp|string, <ts2>  
date|timestamp|string, <roundOff> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: The result is positive if ts1 is later than ts2. If ts1 and ts2 fall on the same day of the month, or if both are the last day of the month, the time of day is ignored. Otherwise, the difference is calculated based on 31 days per month and rounded to 8 decimal places, unless roundOff is false.
- Return Type: double
- Example:

```
> SELECT months_between('1997-02-28 10:30:00', '1996-10-30');  
3.94959677  
> SELECT months_between('1997-02-28 10:30:00', '1996-10-30', false);  
3.9495967741935485
```

## NEXT\_DAY

- Function Syntax:

```
NEXT_DAY(<start_date> date|timestamp|string, <day_of_week> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the first specified weekday after start\_date.
- Return Type:  
SparkSQL: date  
Presto: string
- Example:

```
> SELECT next_day('2015-01-14', 'TU');  
2015-01-20
```



## NOW

- Function Syntax:

```
NOW ()
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the current timestamp.
- Return type: timestamp.
- Example:

```
> SELECT now();  
2020-04-25 15:49:11.914
```

## QUARTER

- Function Syntax:

```
QUARTER(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the quarter in which date d falls.
- Return type: integer.
- Example:

```
> SELECT quarter('2016-08-31');  
3
```

## SECOND

- Function Syntax:

```
SECOND(<ts> timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the second component of the current timestamp.
- Return type: integer.

- Example:

```
> SELECT second('2009-07-30 12:58:59');  
59
```

## TO\_TIMESTAMP

- Function Syntax:

```
TO_TIMESTAMP(<ts_str> string[, <fmt> string]
```

- Supported engines: SparkSQL, Presto.
- Usage: Parses the ts\_str expression in the format fmt into a timestamp. Returns NULL for invalid input. By default, if fmt is omitted, it follows the timestamp coercion rules. The data type of the result matches the configured value.
- Return type: timestamp.
- Example:

```
> SELECT to_timestamp('2016-12-31 00:12:00');  
2016-12-31 00:12:00  
> SELECT to_timestamp('2016-12-31', 'yyyy-MM-dd');  
2016-12-31 00:00:00
```

## TO\_DATE

- Function Syntax:

```
TO_DATE(<date_str> string[, <fmt> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Parses the date\_str expression in the format fmt into a date. Returns NULL for invalid input. By default, if fmt is omitted, it follows the date coercion rules. The data type of the result matches the configured value.
- Return type: date.
- Example:

```
> SELECT to_date('2009-07-30 04:17:52');  
2009-07-30
```

```
> SELECT to_date('2016-12-31', 'yyyy-MM-dd');  
2016-12-31
```

## TO\_UNIX\_TIMESTAMP

- Function Syntax:

```
TO_UNIX_TIMESTAMP(<ts> date|timestamp|string[, <fmt> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the unix timestamp of ts.
- Return type: bigint.
- Example:

```
> SELECT to_unix_timestamp('2016-04-08', 'yyyy-MM-dd');  
1460098800
```

## TO\_UTC\_TIMESTAMP

- Function Syntax:

```
-- SparkSQL  
TO_UTC_TIMESTAMP(<ts> date|timestamp|string, <timezone> string)  
  
-- Presto  
TO_UTC_TIMESTAMP(<ts> date|timestamp|string|interger|double|decimal,  
<timezone> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts a timestamp from a specified time zone to UTC.
- Return type: timestamp.
- Example:

```
-- SparkSQL  
> SELECT to_utc_timestamp('2016-08-31', 'Asia/Seoul');  
2016-08-30 15:00:00  
-- Presto  
> select to_utc_timestamp(10000, 'UTC');
```

```
1970-01-01 08:00:10
```

## TRUNC

- **Function Syntax:**

```
TRUNC(<d> date|string, <fmt> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the date value obtained by truncating date *d* to the time unit specified by *fmt*.
- **Return type:** date.
- **Example:**

```
> SELECT trunc('2019-08-04', 'week');
2019-07-29
> SELECT trunc('2019-08-04', 'quarter');
2019-07-01
> SELECT trunc('2009-02-12', 'MM');
2009-02-01
> SELECT trunc('2015-10-27', 'YEAR');
2015-01-01
```

## DATE\_TRUNC

- **Function Syntax:**

```
DATE_TRUNC(<fmt> string, <ts> date|timestamp|string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the timestamp obtained by truncating timestamp *ts* according to *fmt*.
- **Return type:** timestamp.
- **Example:**

```
> SELECT date_trunc('YEAR', '2015-03-05T09:32:05.359');
2015-01-01 00:00:00
> SELECT date_trunc('MM', '2015-03-05T09:32:05.359');
2015-03-01 00:00:00
```

```
> SELECT date_trunc('DD', '2015-03-05T09:32:05.359');
2015-03-05 00:00:00
> SELECT date_trunc('HOUR', '2015-03-05T09:32:05.359');
2015-03-05 09:00:00
> SELECT date_trunc('MILLISECOND', '2015-03-05T09:32:05.123456');
2015-03-05 09:32:05.123
```

## UNIX\_TIMESTAMP

- Function Syntax:

```
UNIX_TIMESTAMP([<ts> date|timestamp|string[, fmt]])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the UNIX timestamp for the current or a specified time.
- Return type: bigint.
- Example:

```
> SELECT unix_timestamp();
1476884637
> SELECT unix_timestamp('2016-04-08', 'yyyy-MM-dd');
1460041200
```

## DAYOFWEEK

- Function Syntax:

```
DAYOFWEEK(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the day of the week for the date/timestamp "d". The return value ranges from 1 to 7, where:
  - 1 = Sunday (Sunday)
  - 2 = Monday (Monday)
  - 3 = Tuesday (Tuesday)
  - 4 = Wednesday (Wednesday)
  - 5 = Thursday (Thursday)
  - 6 = Friday (Friday)

7 = Saturday (Saturday)

- Return type: integer.
- Example:

```
> SELECT dayofweek('2009-07-30');  
5
```

## WEEKDAY

- Function Syntax:

```
WEEKDAY(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the day of the week for the date/timestamp "d". The return value ranges from 0 to 6, where:
  - 0 = Monday (Monday)
  - 1 = Tuesday (Tuesday)
  - 2 = Wednesday (Wednesday)
  - 3 = Thursday (Thursday)
  - 4 = Friday (Friday)
  - 5 = Saturday (Saturday)
  - 6 = Sunday (Sunday)
- Return type: integer.
- Example:

```
> SELECT weekday('2009-07-30');  
3
```

## WEEKOFYEAR

- Function Syntax:

```
WEEKOFYEAR(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the week number of the year for a given date/timestamp "d".
- Return type: integer.

- Example:

```
> SELECT weekofyear('2008-02-20');  
8
```

## YEAR

- Function Syntax:

```
YEAR(<d> date|timestamp|string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the year from the date/timestamp "d".
- Return type: integer.
- Example:

```
> SELECT year('2016-07-30');  
2016
```

## MAKE\_DATE

- Function Syntax:

```
MAKE_DATE(<year> integer, <month> integer, <day> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Creates a date from the year, month, and day fields.
- Return type: date.
- Example:

```
> SELECT make_date(2013, 7, 15);  
2013-07-15  
> SELECT make_date(2019, 7, NULL);  
NULL
```

## MAKE\_TIMESTAMP

- Function Syntax:

```
MAKE_TIMESTAMP(<year> integer, <month> integer, <day> integer, <hour>
integer, <min> integer, <sec> integer|double|decimal[, <timezone>
string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Creates a timestamp from the given fields.
- Return type: timestamp.
- Example:

```
> SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887);
2014-12-28 06:30:45.887
> SELECT make_timestamp(2014, 12, 28, 6, 30, 45.887, 'CET');
2014-12-27 21:30:45.887
> SELECT make_timestamp(2019, 6, 30, 23, 59, 60);
2019-07-01 00:00:00
> SELECT make_timestamp(2019, 6, 30, 23, 59, 1);
2019-06-30 23:59:01
> SELECT make_timestamp(null, 7, 22, 15, 30, 0);
NULL
```

## DATE\_PART

- Function Syntax:

```
DATE_PART(<field> string, <source> date|timestamp)
```

- Supported engines: SparkSQL, Presto.
- Usage: Extracts a part of a date/timestamp.
- Return type: integer|double.
- Example:

```
> SELECT date_part('YEAR', TIMESTAMP '2019-08-12 01:00:00.123456');
2019
> SELECT date_part('week', timestamp '2019-08-12 01:00:00.123456');
33
> SELECT date_part('doy', DATE '2019-08-12');
224
```



```
> SELECT date_part('SECONDS', timestamp'2019-10-01 00:00:01.000001');  
1.000001
```

## DATE\_FROM\_UNIX\_DATE

- Function Syntax:

```
DATE_FROM_UNIX_DATE(<unix_timestamp> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Creates a date based on the number of days since 1970-01-01.
- Return type: date.
- Example:

```
> SELECT date_from_unix_date(1);  
1970-01-02
```

## UNIX\_DATE

- Function Syntax:

```
UNIX_DATE(<d> date)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the number of days since 1970-01-01.
- Return type: integer.
- Example:

```
> SELECT unix_date(DATE("1970-01-02"));  
1
```

## TIMESTAMP\_SECONDS

- Function Syntax:

```
TIMESTAMP_SECONDS(<sec> bigint|double|decimal)
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Creates a timestamp from the number of seconds since the UTC epoch.
- **Return type:** timestamp.
- **Example:**

```
> SELECT timestamp_seconds(1230219000);  
2008-12-25 07:30:00  
  
> SELECT timestamp_seconds(1230219000.123);  
2008-12-25 07:30:00.123
```

## TIMESTAMP\_MILLIS

- **Function Syntax:**

```
TIMESTAMP_MILLIS(<milli> bigint|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Creates a timestamp from the number of milliseconds since the UTC epoch.
- **Return type:** timestamp.
- **Example:**

```
> SELECT timestamp_millis(1230219000123);  
2008-12-25 07:30:00.123
```

## TIMESTAMP\_MICROS

- **Function Syntax:**

```
TIMESTAMP_MICROS(<micro> bigint)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Creates a timestamp from the number of milliseconds since the UTC epoch.
- **Return type:** timestamp.
- **Example:**

```
> SELECT timestamp_micros(1230219000123123);  
2008-12-25 07:30:00.123123
```

## UNIX\_SECONDS

- Function Syntax:

```
UNIX_SECONDS(<ts> timestamp)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the number of seconds since 1970-01-01 00:00:00 UTC.
- Return type: bigint.
- Example:

```
> SELECT unix_seconds(TIMESTAMP('1970-01-01 00:00:01Z'));  
1
```

## UNIX\_MILLIS

- Function Syntax:

```
UNIX_MILLIS(<ts> timestamp)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the number of milliseconds since 1970-01-01 00:00:00 UTC.
- Return type: bigint.
- Example:

```
> SELECT unix_millis(TIMESTAMP('1970-01-01 00:00:01Z'));  
1000
```

## UNIX\_MICROS

- Function Syntax:

```
UNIX_MICROS(<ts> timestamp)
```

- Supported engines: SparkSQL.
- Usage: Returns the number of microseconds since 1970-01-01 00:00:00 UTC.
- Return type: bigint.

- Example:

```
> SELECT unix_micros(timestamp '1970 00:00:00.001')
1000
```

## DATEADD/TIMESTAMP\_ADD

- Function Syntax:

```
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
```

- Supported engines: SparkSQL.
- Engine version requirement: The engine must be purchased on or after December 7, 2023, or upgraded to the latest version.
- Usage: Calculates the final time based on a specified date or time, a specified unit, and an offset.
  - dt: The input date or time format. Returns null if it is null.
  - delta: Required. The data type is int. Returns null if it is null.
  - pattern: Required. The data type is string. The unit is pattern. The following patterns are supported:
    - Year (standard "y", compatible with "-year", "yyyy")
    - Month (standard "M", compatible with "-month", "-mon", "MM")
    - Day (standard "d", compatible with "-day", "dd")
    - Hour (standard "H", compatible with "-hour", "hh", "HH", "h")
    - Minute (standard "m", compatible with "mi", "mm")
    - Second (standard "s", compatible with "ss")
    - Microsecond (standard "S", compatible with "SSS")

Input of any other type is not allowed and will cause an error.

### Note:

- When the unit of delta is month, if the month part of delta does not cause a Day overflow after the delta value is added, the Day value remains unchanged. Otherwise, the Day value is set to the last day of the resulting month.
- Note: The output time is timezone-aware by default.
- Note the distinction between the month unit "M" and the minute unit "m". The system supports both the 0–12 hour format "h" and the 0–24 hour format "H", but

using "H" is recommended.

- Return type: timestamp.
- Example:

```
> SELECT timestamp_add(date '2016-12-31', -1, 'M');  
2016-11-30T00:00:00.000+08:00  
> SELECT timestamp_add(timestamp '2016-12-31 00:12:00', 1, 'm');  
2016-12-31T00:13:00.000+08:00  
> SELECT timestamp_add(timestamp '2016-12-31 00:00:00', -1, 'm');  
2016-12-30T23:59:00.000+08:00
```

# JSON Functions

Last updated: 2026-05-21 14:32:06

## GET\_JSON\_OBJECT

- Function Syntax:

```
GET_JSON_OBJECT(<json> string, <path> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Extract a json object.
- Return type: string.
- Example:

```
> SELECT get_json_object('{\"a\":\"b\"}', '$.a');  
b
```

## JSON\_TUPLE

- Function Syntax:

```
JSON_TUPLE(<json> string, <p1> string, ..., <pn> string)
```

- Supported engines: SparkSQL.
- Usage: This function returns a tuple similar to that of the `get\_json\_object` function, but it requires multiple names as input. All input parameters and output column types are strings.
- Return type: struct<string, ..., string>.
- Example:

```
> SELECT json_tuple('{\"a\":1, \"b\":2}', 'a', 'b');  
1 2
```

## TO\_JSON

- Function Syntax:

```
TO_JSON(<expr> struct|map|array[, <option> map])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a JSON string with the specified structure value.
- Return type: string.
- Example:

```
> SELECT to_json(named_struct('a', 1, 'b', 2));
{"a":1,"b":2}
> SELECT to_json(named_struct('time', to_timestamp('2015-08-26',
'yyyy-MM-dd')), map('timestampFormat', 'dd/MM/yyyy'));
{"time":"26/08/2015"}
> SELECT to_json(array(named_struct('a', 1, 'b', 2)));
[{"a":1,"b":2}]
> SELECT to_json(map('a', named_struct('b', 1)));
{"a":{"b":1}}
> SELECT to_json(map(named_struct('a', 1), named_struct('b', 2)));
{"[1]":{"b":2}}
> SELECT to_json(map('a', 1));
{"a":1}
> SELECT to_json(array((map('a', 1))));
[{"a":1}]
```

## FROM\_JSON

- Function Syntax:

```
FROM_JSON(<json> string, <schema> string[, <options> map<string,
string>])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a structured value with the given jsonStr and schema.
- Return type: struct.
- Example:

```
> SELECT from_json('{ "a":1, "b":0.8}', 'a INT, b DOUBLE');
{"a":1,"b":0.8}
```

```
> SELECT from_json('{"time":"26/08/2015"}', 'time Timestamp',  
map('timestampFormat', 'dd/MM/yyyy'));  
{"time":2015-08-26 00:00:00}
```

## SCHEMA\_OF\_JSON

- Function Syntax:

```
SCHEMA_OF_JSON(<json> string[, <options> map<string, string>])
```

- Supported engines: SparkSQL.
- Usage: Returns a structure in DDL format for a JSON string.
- Return type: string.
- Example:

```
> SELECT schema_of_json(' [{"col":0}] ');  
ARRAY<STRUCT<`col`: BIGINT>>  
> SELECT schema_of_json(' [{"col":01}] ',  
map('allowNumericLeadingZeros', 'true'));  
ARRAY<STRUCT<`col`: BIGINT>>
```

## JSON\_ARRAY\_LENGTH

- Function Syntax:

```
JSON_ARRAY_LENGTH(<jsonArray> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the number of elements in the outermost JSON array.
- Return type: integer.
- Example:

```
> SELECT json_array_length('[1,2,3,4]');  
4  
> SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');  
5  
> SELECT json_array_length('[1,2]');
```



```
NULL
```

## JSON\_OBJECT\_KEYS

- **Function Syntax:**

```
JSON_OBJECT_KEYS(<json> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns all keys of the outermost JSON object in an array.
- **Return type:** array<string>.
- **Example:**

```
> SELECT json_object_keys('{}');  
[]  
> SELECT json_object_keys('{"key": "value"}');  
["key"]  
> SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}');  
["f1","f2"]
```

# Math Function

Last updated: 2026-05-21 14:33:34

## ABS

- Function Syntax:

```
ABS(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the absolute value of expr.
- Return Type: Same as <expr>.
- Example:

```
> SELECT abs(-1);  
1
```

## ACOS

- Function Syntax:

```
ACOS(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the arccosine of expr.
- Return Type: double
- Example:

```
> SELECT acos(1);  
0.0  
> SELECT acos(2);  
NaN
```

## ACOSH

- Function Syntax:

```
ACOSH(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the inverse hyperbolic cosine of expr.
- Return Type: double
- Example:

```
> SELECT acosh(1);  
0.0  
> SELECT acosh(0);  
NaN
```

## ASIN

- Function Syntax:

```
ASIN(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the arcsine of expr.
- Return Type: double
- Example:

```
> SELECT asin(0);  
0.0  
> SELECT asin(2);  
NaN
```

## ASINH

- Function Syntax:

```
ASINH(<expr> integer|double|decimal)
```

- Supported engines: SparkSQL.
- Usage: Returns the inverse hyperbolic sine of expr.
- Return Type: double

- Example:

```
> SELECT asinh(0);  
0.0
```

## ATAN

- Function Syntax:

```
ATAN(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the arctangent of expr.
- Return Type: double
- Example:

```
> SELECT atan(0);  
0.0
```

## ATAN2

- Function Syntax:

```
ATAN2(<x>, integer|double|decimal, <y> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the radian angle between the positive x-axis of the plane and the point specified by the coordinates.
- Return Type: double
- Example:

```
> SELECT atan2(0, 0);  
0.0
```

## ATANH

- Function Syntax:

ATANH (<expr> integer|double|decimal)

- Supported Engines: SparkSQL, Presto
- Usage: Returns the inverse hyperbolic tangent of expr.
- Return Type: double
- Example:

```
> SELECT atanh(0);
0.0
> SELECT atanh(2);
NaN
```

# BIN

- **Function Syntax:**

```
BIN(<expr> integer)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the string representation of the long value `expr` expressed in binary.
- **Return type:** string
- **Example:**

# BROUND

- **Function Syntax:**

```
BROUND(<expr> integer|double|decimal, <d> integer)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns `expr` rounded to `d` decimal places using the `HALF_EVEN` rounding mode.
- **Return Type:** decimal

- Example:

```
> SELECT round(2.5, 0);  
2
```

## CBRT

- Function Syntax:

```
CBRT(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the cube root of expr.
- Return Type: double
- Example:

```
> SELECT cbrt(27.0);  
3.0
```

## CEIL

- Function Syntax:

```
CEIL(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the smallest integer that is not less than expr.
- Return Type: integer
- Example:

```
> SELECT ceil(-0.1);  
0  
> SELECT ceil(5);  
5
```

## COS

- Function Syntax:

```
COS(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the cosine of expr.
- Return Type: double
- Example:

```
> SELECT cos(0);  
1.0
```

## COSH

- Function Syntax:

```
COSH(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the hyperbolic cosine of expr.
- Return Type: double
- Example:

```
> SELECT cosh(0);  
1.0
```

## CONV

- Function Syntax:

```
CONV(<num> bigint|string, <from_base> integer, <to_base> integer)
```

- Supported Engines: SparkSQL, Presto
- Usage: Converts num from from\_base to to\_base.
- Return type: string
- Example:

```
> SELECT conv('100', 2, 10);
```

```
4
> SELECT conv(-10, 16, -10);
-16
```

## DEGREES

- Function Syntax:

```
DEGREES(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Converts radians to degrees.
- Return Type: double
- Example:

```
> SELECT degrees(3.141592653589793);
180.0
```

## E

- Function Syntax:

```
E()
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns Euler's constant.
- Return Type: double
- Example:

```
> SELECT e();
2.718281828459045
```

## EXP

- Function Syntax:

```
EXP(<expr> integer|double|decimal)
```



- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns Euler's constant  $e$  raised to the power of `expr`.
- **Return Type:** double
- **Example:**

```
> SELECT exp(0);  
1.0
```

## EXPM1

- **Function Syntax:**

```
EXPM1(<expr> integer|double|decimal)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns  $\text{EXP}(\text{expr}) - 1$ .
- **Return Type:** double
- **Example:**

```
> SELECT expm1(0);  
0.0
```

## FLOOR

- **Function Syntax:**

```
FLOOR(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the largest integer that is not greater than `expr`.
- **Return Type:** integer
- **Example:**

```
> SELECT floor(-0.1);  
-1  
  
> SELECT floor(5);  
5
```

## FACTORIAL

- Function Syntax:

```
FACTORIAL(<expr> integer)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the factorial of expr. The value of expr must be within the range [0..20]. Otherwise, NULL is returned.
- Return type: bigint
- Example:

```
> SELECT factorial(5);  
120
```

## HEX

- Function Syntax:

```
HEX(<expr> bigint|string)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns expr in hexadecimal.
- Return type: string
- Example:

```
> SELECT hex(17);  
11  
> SELECT hex('Spark SQL');  
537061726B2053514C
```

## HYPOT

- Function Syntax:

```
HYPOT(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto

- **Usage:** Returns  $\text{sqrt}(\text{pow}(\text{expr1}, 2) + \text{pow}(\text{expr2}, 2))$ .
- **Return Type:** double
- **Example:**

```
> SELECT hypot(3, 4);  
5.0
```

## LOG

- **Function Syntax:**

```
LOG(<base> integer|double|decimal, <expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the logarithm of expr to the base base.
- **Return Type:** double
- **Example:**

```
> SELECT log(10, 100);  
2.0
```

## LOG10

- **Function Syntax:**

```
LOG10(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the base-10 logarithm of expr.
- **Return Type:** double
- **Example:**

```
> SELECT log10(10);  
1.0
```

## LOG1P

- **Function Syntax:**

```
LOG1P(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns  $\log(1 + \text{expr})$ .
- **Return Type:** double
- **Example:**

```
> SELECT log1p(0);  
0.0
```

## LOG2

- **Function Syntax:**

```
LOG2(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the base-2 logarithm of  $\text{expr}$ .
- **Return Type:** double
- **Example:**

```
> SELECT log2(2);  
1.0
```

## LN

- **Function Syntax:**

```
LN(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the natural logarithm of  $\text{expr}$  (base  $e$ ).
- **Return Type:** double
- **Example:**

```
> SELECT ln(1);  
0.0
```

## MOD

- **Function Syntax:**

```
MOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)  
<expr1> MOD <expr2>
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the remainder after expr1 is divided by expr2.
- **Return Type:** double|integer
- **Example:**

```
> SELECT 2 % 1.8;  
0.2  
> SELECT MOD(2, 1.8);  
0.2
```

## NEGATIVE

- **Function Syntax:**

```
NEGATIVE(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the negative value of expr.
- **Return Type:** Same as expr.
- **Example:**

```
> SELECT negative(1);  
-1
```

## PI

- **Function Syntax:**

```
PI()
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns PI.
- Return Type: double
- Example:

```
> SELECT pi();  
3.141592653589793
```

## PMOD

- Function Syntax:

```
PMOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the positive value of `expr1 mod expr2`.
- Return Type: double
- Example:

```
> SELECT pmod(10, 3);  
1  
> SELECT pmod(-10, 3);  
2
```

## POSITIVE

- Function Syntax:

```
POSITIVE(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns `expr`.
- Return Type: Same as `expr`.
- Example:

```
> SELECT positive(1);  
1
```

## POWER

- Function Syntax:

```
POWER(<base> integer|double|decimal, <number> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns base raised to the power of number.
- Return Type: double
- Example:

```
> SELECT power(2, 3);  
8.0
```

## POW

- Function Syntax:

```
POW(<base> integer|double|decimal, <number> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns base raised to the power of number.
- Return Type: double
- Example:

```
> SELECT pow(2, 3);  
8.0
```

## RADIANS

- Function Syntax:

```
RADIANS(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Converts degrees to radians.
- **Return Type:** double
- **Example:**

```
> SELECT radians(180);  
3.141592653589793
```

## RINT

- **Function Syntax:**

```
RINT(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the double-precision value that is the nearest to the parameter value and equals a mathematical integer.
- **Return Type:** double
- **Example:**

```
> SELECT rint(12.3456);  
12.0
```

## ROUND

- **Function Syntax:**

```
ROUND(<expr> integer|double|decimal, <d> integer)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Rounds expr to d decimal places using the half-up rounding mode.
- **Return Type:** double
- **Example:**

```
> SELECT round(2.5, 0);  
3
```



## SHIFLEFT

- Function Syntax:

```
SHIFLEFT(<base> integer|double|decimal, <expr> integer)
```

- Supported Engines: SparkSQL, Presto
- Usage: Performs a bitwise left shift.
- Return Type: int | bigint
- Example:

```
> SELECT shifleft(2, 1);  
4
```

## SHIFTRIGHT

- Function Syntax:

```
SHIFTRIGHT(<base> integer|double|decimal, <expr> integer)
```

- Supported Engines: SparkSQL, Presto
- Usage: Performs a bitwise right shift.
- Return Type: int | bigint
- Example:

```
> SELECT shiftright(4, 1);  
2
```

## SHIFTRIGHTUNSIGNED

- Function Syntax:

```
SHIFTRIGHTUNSIGNED(<base> integer|double|decimal, <expr> integer)
```

- Supported Engines: SparkSQL, Presto
- Usage: Performs an unsigned bitwise right shift.
- Return Type: int | bigint

- Example:

```
> SELECT shiftrightunsigned(4, 1);  
2
```

## SIGN

- Function Syntax:

```
SIGN(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns -1.0, 0.0, or 1.0 when expr is negative, zero, or positive, respectively.
- Return Type: double
- Example:

```
> SELECT sign(40);  
1.0
```

## SIGNUM

- Function Syntax:

```
SIGNUM(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns -1.0, 0.0, or 1.0 when expr is negative, zero, or positive, respectively.
- Return Type: double
- Example:

```
> SELECT signum(40);  
1.0
```

## SIN

- Function Syntax:

```
SIN(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the sine of expr.
- Return Type: double
- Example:

```
> SELECT sin(0);  
0.0
```

## SINH

- Function Syntax:

```
SINH(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the hyperbolic sine of expr.
- Return Type: double
- Example:

```
> SELECT sinh(0);  
0.0
```

## SQRT

- Function Syntax:

```
SQRT(<expr> integer|double|decimal)
```

- Supported Engines: SparkSQL, Presto
- Usage: Returns the square root of expr.
- Return Type: double
- Example:

```
> SELECT sqrt(4);
```

```
2.0
```

## TAN

- **Function Syntax:**

```
TAN(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the tangent of expr.
- **Return Type:** double
- **Example:**

```
> SELECT tan(0);  
0.0
```

## COT

- **Function Syntax:**

```
COT(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns the cotangent of the expression.
- **Return Type:** double
- **Example:**

```
> SELECT cot(1);  
0.6420926159343306
```

## TANH

- **Function Syntax:**

```
TANH(<expr> integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto

- **Usage:** Returns the hyperbolic tangent of expr.
- **Return Type:** double
- **Example:**

```
> SELECT tanh(0);  
0.0
```

## WIDTH\_BUCKET

- **Function Syntax:**

```
WIDTH_BUCKET(<value> integer|double|decimal, <min>  
integer|double|decimal, <max> integer|double|decimal, <num_bucket>  
integer)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Divides the range from min to max into num\_bucket equal groups and returns the group number into which the value falls.
- **Return Type:** integer
- **Example:**

```
> SELECT width_bucket(5.3, 0.2, 10.6, 5);  
3  
> SELECT width_bucket(-2.1, 1.3, 3.4, 3);  
0  
> SELECT width_bucket(8.1, 0.0, 5.7, 4);  
5  
> SELECT width_bucket(-0.9, 5.2, 0.5, 2);  
3
```

## TRY\_ADD

- **Function Syntax:**

```
TRY_ADD(<expr1> integer|double|decimal|date|timestamp, <expr2>  
integer|double|decimal|date|timestamp)
```

- **Supported Engines:** SparkSQL, Presto

- **Usage:** Returns the sum of expr1 and expr2. The result is null on overflow.
- **Return Type:** integer|double|decimal|date|timestamp
- **Example:**

```
> SELECT try_add(1, 2);
3
> SELECT try_add(2147483647, 1);
NULL
> SELECT try_add(date'2021-01-01', 1);
2021-01-02
```

## TRY\_DIVIDE

- **Function Syntax:**

```
TRY_DIVIDE(<dividend> integer|double|decimal, <divisor>
integer|double|decimal)
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns dividend/divisor.
- **Return Type:** double
- **Example:**

```
> SELECT try_divide(3, 2);
1.5
> SELECT try_divide(2L, 2L);
1.0
> SELECT try_divide(1, 0);
NULL
```

## RAND

- **Function Syntax:**

```
RAND([<seed> integer])
```

- **Supported Engines:** SparkSQL, Presto

- **Usage:** Returns a random value. This value is independently and uniformly distributed within the range [0, 1].
- **Return Type:** double
- **Example:**

```
> SELECT rand();
0.9629742951434543
> SELECT rand(0);
0.7604953758285915
> SELECT rand(null);
0.7604953758285915
```

## RANDOM

- **Function Syntax:**

```
RANDOM([<seed> integer])
```

- **Supported Engines:** SparkSQL, Presto
- **Usage:** Returns a random value. This value is independently and uniformly distributed within the range [0, 1].
- **Return Type:** double
- **Example:**

```
> SELECT rand();
0.9629742951434543
> SELECT rand(0);
0.7604953758285915
> SELECT rand(null);
0.7604953758285915
```

## RANDN

- **Function Syntax:**

```
RANDN([<seed> integer])
```

- **Supported Engines:** SparkSQL, Presto

- **Usage:** Returns a random value containing independent and identically distributed (i.i.d.) values drawn from a standard normal distribution. The implementation of this function differs between SPARKSQL and PRESTO. Using the same seed may yield different results.
- **Return Type:** double
- **Example:**

```
> SELECT randn();
-0.3254147983080288
> SELECT randn(0);
1.6034991609278433
> SELECT randn(null);
1.6034991609278433
```

## DIV

- **Function Syntax:**

```
<expr1> DIV <expr2>
```

- **Supported engines:** SparkSQL.
- **Usage:** Divides expr1 by expr2.
- **Return Type:** integer
- **Example:**

```
> SELECT 3 div 2;
1
```



# String Functions

Last updated: 2026-05-21 14:34:09

## ASCII

- Function Syntax:

```
ASCII(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the numeric value of the first character in str.
- Return type: integer.
- Example:

```
> SELECT ascii('222');  
50
```

## BASE64

- Function Syntax:

```
-- SparkSQL  
BASE64(<str> string|binary)  
  
-- Presto  
BASE64(<str> binary)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts the parameter to a Base64 string.
- Return type: string.
- Example:

```
> SELECT base64('tencent');  
dGVuY2VudA==
```

## BIT\_LENGTH

- Function Syntax:

```
BIT_LENGTH(<expr> string|binary)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the bit length of string data or the number of bits in binary data.
- Return type: integer.
- Example:

```
> select bit_length('tencent');  
56  
> select bit_length(binary('tencent'));  
56
```

## CHAR

- Function Syntax:

```
CHAR(<expr> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the ASCII character of expr. If expr is greater than 256, expr is set to expr % 256.
- Return type: string.
- Example:

```
> SELECT char(65);  
A
```

## CHR

- Function Syntax:

```
CHR(<expr> integer)
```

- Supported engines: SparkSQL.
- Usage: Returns the ASCII character of expr. If expr is greater than 256, expr is set to expr % 256.
- Return type: string.

- Example:

```
> SELECT chr(65);  
A
```

## CHAR\_LENGTH

- Function Syntax:

```
CHAR_LENGTH(<expr> string|binary)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the character length of string data or the number of bytes of binary data. The length of string data includes trailing spaces. The length of binary data includes binary zeros.
- Return type: integer.
- Example:

```
> select char_length(binary('tencent'));  
7  
> select char_length('tencent');  
7
```

## CHARACTER\_LENGTH

- Function Syntax:

```
CHARACTER_LENGTH(<expr> string|binary)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the character length of string data or the number of bytes of binary data. The length of string data includes trailing spaces. The length of binary data includes binary zeros.
- Return type: integer.
- Example:

```
> select character_length(binary('tencent'));  
7
```

```
> select character_length('tencent');  
7
```

## CONCAT\_WS

- Function Syntax:

```
CONCAT_WS(<sep> string[, <s> string|array<string>]]+)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a string separated by sep.
- Return type: string.
- Example:

```
> SELECT concat_ws(' ', 'tencent', 'dlc');  
tencent dlc
```

## DECODE

- Function Syntax:

```
DECODE(<expr> binary|string, <charset>string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Decodes the first parameter using the character set of the second parameter.
- Return type: string.
- Example:

```
> SELECT decode(encode('abc', 'utf-8'), 'utf-8');  
abc
```

## ELT

- Function Syntax:

```
ELT(<n> integer, <s1> string, <s2> string, ...)
```

- Supported engines: SparkSQL, Presto.

- Usage: Returns the nth element.
- Return type: string.
- Example:

```
> SELECT elt(1, 'scala', 'java');  
scala
```

## ENCODE

- Function Syntax:

```
ENCODE(<expr> binary|string, <charset> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Encodes the first parameter using the character set of the second parameter.
- Return type: string.
- Example:

```
> SELECT encode('abc', 'utf-8');  
abc  
> SELECT encode(x'616263', 'utf-8');  
abc
```

## FIND\_IN\_SET

- Function Syntax:

```
FIND_IN_SET(<str> string, <str_array> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the index (starting from 1) of the given string str in the comma-separated list atr\_array. If the string is not found or str contains a comma, 0 is returned.
- Return type: integer.
- Example:

```
> SELECT find_in_set('ab', 'abc,b,ab,c,def');  
3
```

## FORMAT\_NUMBER

- Function Syntax:

```
FORMAT_NUMBER(<expr1> integer|double|decimal, <expr2> string|integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Formats expr1 as '#,####,####.##' and rounds it to expr2 decimal places. If expr2 is 0, the result has no decimal point or fractional part.
- Return type: string.
- Example:

```
> SELECT format_number(12332.123456, 4);  
12,332.1235  
  
> SELECT format_number(12332.123456, '#####.###');  
12332.123
```

## FORMAT\_STRING

- Function Syntax:

```
FORMAT_STRING(<str> string, obj <T>, ...)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the formatted string from a printf-style format string.
- Return type: string.
- Example:

```
> SELECT format_string("Hello World %d %s", 100, "days");  
Hello World 100 days
```

## INITCAP

- Function Syntax:

```
INITCAP(<str> string)
```

- Supported engines: SparkSQL, Presto.

- Usage: Capitalizes the first letter of each word and converts all other letters to lowercase.
- Return type: string.
- Example:

```
> SELECT initcap('sPark sql');  
Spark Sql
```

## INSTR

- Function Syntax:

```
INSTR(<str> string, <substr> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the index (starting from 1) of the first occurrence of substr in str.
- Return type: integer.
- Example:

```
> SELECT instr('SparkSQL', 'SQL');  
6
```

## LCASE

- Function Syntax:

```
LCASE(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts str to lowercase.
- Return type: string.
- Example:

```
> SELECT lcase('SparkSQL');  
sparksql
```

## LENGTH

- **Function Syntax:**

```
LENGTH(<expr> string|binary)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the character length of string data or the number of bytes of binary data. The length of string data includes trailing spaces. The length of binary data includes binary zeros.
- **Return type:** integer.
- **Example:**

```
> SELECT length('Spark SQL ');  
10
```

## LEVENSHTEIN

- **Function Syntax:**

```
LEVENSHTEIN(<s1> string, <s2> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the Levenshtein distance between two given strings.
- **Return type:** integer.
- **Example:**

```
> SELECT levenshtein('kitten', 'sitting');  
3
```

## LIKE

- **Function Syntax:**

```
LIKE(<s1> str, <s2> pattern)  
<str> like <pattern>[ ESCAPE <escape>]
```

- **Supported engines:** SparkSQL, Presto.



- **Usage:** Returns true if str matches the pattern with escape; returns null if any parameter is null; otherwise, returns false.
- **Return type:** boolean.
- **Example:**

```
> SELECT like('Spark', '_park');
true
> SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
> SELECT '%SystemDrive%\Users\John' like '%SystemDrive%\%\Users%';
true
> SET spark.sql.parser.escapedStringLiterals=false;
spark.sql.parser.escapedStringLiterals  false
> SELECT '%SystemDrive%\Users\John' like
'%SystemDrive%\%\Users%';
false
> SELECT '%SystemDrive%/Users/John' like '%SystemDrive%///Users%'
ESCAPE '/';
true
```

## LOWER

- **Function Syntax:**

```
LOWER(<str> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns str with all characters converted to lowercase.
- **Return type:** string.
- **Example:**

```
> SELECT lower('TENCENT');
tencent
```

## LOCATE

- **Function Syntax:**

```
LOCATE(<substr> string, <str> string[, <pos> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the position of the first occurrence of substr in str after the pos-th character.
- Return type: integer.
- Example:

```
> SELECT locate('bar', 'foobarbar');
4
> SELECT locate('bar', 'foobarbar', 5);
7
```

## OCTET\_LENGTH

- Function Syntax:

```
OCTET_LENGTH(<expr> string|binary)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the byte length of string data or the number of bytes in binary data.
- Return type: integer.
- Example:

```
> SELECT octet_length('Spark SQL');
9
```

## LPAD

- Function Syntax:

```
LPAD(<str> string, <len> integer[, <pad> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns str, left-padded to length len with pad. If str is longer than len, the return value is truncated to len characters. If pad is not specified, str is padded with spaces.
- Return type: string.

- Example:

```
> SELECT lpad('hi', 5, '??');
???hi
> SELECT lpad('hi', 1, '??');
h
> SELECT lpad('hi', 5);
hi
```

## LTRIM

- Function Syntax:

```
LTRIM(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Removes leading space characters from str.
- Return type: string.
- Example:

```
> SELECT ltrim('    SparkSQL    ');
SparkSQL
```

## PARSE\_URL

- Function Syntax:

```
PARSE_URL(<url> string, <path> string[, <key> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Extracts the path from a url.
- Return type: string.
- Example:

```
> SELECT parse_url('http://spark.apache.org/path?query=1', 'HOST');
spark.apache.org
> SELECT parse_url('http://spark.apache.org/path?query=1', 'QUERY');
query=1
```

```
> SELECT parse_url('http://spark.apache.org/path?query=1', 'QUERY',  
'query');  
1
```

## POSITION

- **Function Syntax:**

```
POSITION(<substr> string, <str> string[, <pos> integer])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the position of the first occurrence of substr in str after the pos-th character.
- **Return type:** integer.
- **Example:**

```
> SELECT position('bar', 'foobarbar');  
4  
> SELECT position('bar', 'foobarbar', 5);  
7  
> SELECT POSITION('bar' IN 'foobarbar');  
4
```

## PRINTF

- **Function Syntax:**

```
PRINTF(<str> string, obj <T>, ...)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the formatted string from a printf-style format string.
- **Return type:** string.
- **Example:**

```
> SELECT printf("Hello World %d %s", 100, "days");  
Hello World 100 days
```

## REPEAT

- Function Syntax:

```
REPEAT(<str> string, <n> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a string that repeats the given string n times.
- Return type: string.
- Example:

```
> SELECT repeat('123', 2);  
123123
```

## REPLACE

- Function Syntax:

```
REPLACE(<str> string, <search> string[, <replace> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Replaces all occurrences of search in str with replace.
- Return type: string.
- Example:

```
> SELECT replace('ABCabc', 'abc', 'DEF');  
ABCDEF
```

## OVERLAY

- Function Syntax:

```
OVERLAY(<input> string, <replace> string, <pos> integer[, <len>  
integer])
```

- Supported engines: SparkSQL.
- Usage: Replaces the substring of input starting at position pos with a length of len with replace.

- Return type: string.
- Example:

```
> SELECT overlay('Spark SQL' PLACING '_' FROM 6);
Spark_SQL
> SELECT overlay('Spark SQL' PLACING 'CORE' FROM 7);
Spark CORE
> SELECT overlay('Spark SQL' PLACING 'ANSI ' FROM 7 FOR 0);
Spark ANSI SQL
> SELECT overlay('Spark SQL' PLACING 'structured' FROM 2 FOR 4);
Structured SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('_',
'utf-8') FROM 6);
Spark_SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('CORE',
'utf-8') FROM 7);
Spark CORE
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING encode('ANSI ',
'utf-8') FROM 7 FOR 0);
Spark ANSI SQL
> SELECT overlay(encode('Spark SQL', 'utf-8') PLACING
encode('structured', 'utf-8') FROM 2 FOR 4);
Structured SQL
```

## RPAD

- Function Syntax:

```
RPAD(<str> string, <len> integer[, <pad> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns str, right-padded to length len with pad. If str is longer than len, the return value is truncated to len characters. If pad is not specified, str is padded with spaces.
- Return type: string.
- Example:

```
> SELECT rpad('hi', 5, '???');
hi???
```

```
> SELECT rpad('hi', 1, '??');  
h  
> SELECT rpad('hi', 5);  
hi
```

## RTRIM

- Function Syntax:

```
RTRIM(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Removes trailing space characters from str.
- Return type: string.
- Example:

```
> SELECT rtrim('      SparkSQL      ');  
SparkSQL
```

## SENTENCES

- Function Syntax:

```
SENTENCES(<str> string[, <lang> string, <country> string])
```

- Supported engines: SparkSQL, Presto.
- Usage: Splits str into an array of words.
- Return type: array<string>.
- Example:

```
> SELECT sentences('Hi there! Good morning.');
```

```
[["Hi", "there"], ["Good", "morning"]]
```

## SOUNDEX

- Function Syntax:

```
SOUNDEX(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the Soundex code of a string.
- Return type: string.
- Example:

```
> SELECT soundex('Miller');  
M460
```

## SPACE

- Function Syntax:

```
SPACE(<n> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a string consisting of n spaces.
- Return type: string.
- Example:

```
> SELECT concat(space(2), '1');  
1
```

## SPLIT

- Function Syntax:

```
SPLIT(<str> string, <regex> string, <limit> integer)
```

- Supported engines: SparkSQL, Presto.
- Usage: Splits str using the string that matches regex as a delimiter and returns an array with a maximum length of limit.
- Return type: array<string>.
- Example:



```
> SELECT split('oneAtwoBthreeC', '[ABC]');
["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', -1);
["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', 2);
["one","twoBthreeC"]
```

## SUBSTRING

- **Function Syntax:**

```
SUBSTRING(<str> string, <pos> integer[, <len> integer])
SUBSTRING(<str> FROM <pos>[ FOR <len>])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a substring of str starting at position pos with a length of len, or a slice of a byte array starting at position pos with a length of len.
- **Return type:** string.
- **Example:**

```
> SELECT substring('Spark SQL', 5);
k SQL
> SELECT substring('Spark SQL', -3);
SQL
> SELECT substring('Spark SQL', 5, 1);
k
> SELECT substring('Spark SQL' FROM 5);
k SQL
> SELECT substring('Spark SQL' FROM -3);
SQL
> SELECT substring('Spark SQL' FROM 5 FOR 1);
k
```

## SUBSTR

- **Function Syntax:**

```
SUBSTR(<str> string, <pos> integer[, <len> integer])
```

```
SUBSTR(<str> FROM <pos>[ FOR <len>])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a substring of str starting at position pos with a length of len, or a slice of a byte array starting at position pos with a length of len.
- **Return type:** string.
- **Example:**

```
> SELECT substr('Spark SQL', 5);  
k SQL  
> SELECT substr('Spark SQL', -3);  
SQL  
> SELECT substr('Spark SQL', 5, 1);  
k  
> SELECT substr('Spark SQL' FROM 5);  
k SQL  
> SELECT substr('Spark SQL' FROM -3);  
SQL  
> SELECT substr('Spark SQL' FROM 5 FOR 1);  
k
```

## LEFT

- **Function Syntax:**

```
LEFT(<str> string, <len> integer)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the leftmost len characters of the string str. If len is less than or equal to 0, the result is an empty string.
- **Return type:** string.
- **Example:**

```
> SELECT left('tencent', 3);  
ten
```

## RIGHT

- **Function Syntax:**

```
RIGHT(<str> string, <len> integer)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the rightmost len characters of the string str. If len is less than or equal to 0, the result is an empty string.
- **Return type:** string.
- **Example:**

```
> SELECT right('tencent', 3);  
ent
```

## SUBSTRING\_INDEX

- **Function Syntax:**

```
SUBSTRING_INDEX(<str> string, <delim> string, <count> integer)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a substring from str before the count-th occurrence of delim. If count is positive, returns everything to the left of the last delimiter (counting from the left). If count is negative, returns everything to the right of the final delimiter (counting from the right). This function is case-sensitive when matching delim.
- **Return type:** string.
- **Example:**

```
> SELECT substring_index('cloud.tencent.com', '.', 2);  
cloud.tencent
```

## TRANSLATE

- **Function Syntax:**

```
TRANSLATE(<input> string, <from> string, <to> string)
```

- **Supported engines:** SparkSQL.

- **Usage:** Transforms the input string by replacing characters in the from string with the corresponding characters in the to string.
- **Return type:** string.
- **Example:**

```
> SELECT translate('AaBbCc', 'abc', '123');  
A1B2C3
```

## TRIM

- **Function Syntax:**

```
TRIM(<str> string)  
trim(BOTH FROM str)  
trim(LEADING FROM str)  
trim(TRAILING FROM str)  
trim(trimStr FROM str)  
trim(BOTH trimStr FROM str)  
trim(LEADING trimStr FROM str)  
trim(TRAILING trimStr FROM str)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:**
  - trim(str) – Removes leading and trailing space characters from str.
  - trim(BOTH FROM str) : Removes leading and trailing space characters from str.
  - trim(LEADING FROM str) : Removes leading space characters from str.
  - trim(TRAILING FROM str) : Removes trailing space characters from str.
  - trim(trimStr FROM str) : Removes the trimStr characters from the beginning and end of str.
  - trim(BOTH trimStr FROM str) : Removes the trimStr characters from the beginning and end of str.
  - trim(LEADING trimStr FROM str) : Removes leading trimStr characters from str.
  - trim(TRAILING trimStr FROM str) : Removes trailing trimStr characters from str.
- **Return type:** string
- **Example:**

```
> SELECT trim('    SparkSQL    ');  
SparkSQL  
> SELECT trim(BOTH FROM '    SparkSQL    ');
```

```

SparkSQL
> SELECT trim(LEADING FROM '      SparkSQL      ');
SparkSQL
> SELECT trim(TRAILING FROM '      SparkSQL      ');
SparkSQL
> SELECT trim('SL' FROM 'SSparkSQLS');
parkSQ
> SELECT trim(BOTH 'SL' FROM 'SSparkSQLS');
parkSQ
> SELECT trim(LEADING 'SL' FROM 'SSparkSQLS');
parkSQLS
> SELECT trim(TRAILING 'SL' FROM 'SSparkSQLS');
SSparkSQ

```

## BTRIM

- **Function Syntax:**

```
BTRIM(<str> string[, <trimStr> string])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Removes the trimStr characters (space by default) from the beginning and end of str.
- **Return type:** string
- **Example:**

```

> SELECT btrim('      SparkSQL      ');
SparkSQL
> SELECT btrim(encode('      SparkSQL      ', 'utf-8'));
SparkSQL
> SELECT btrim('SSparkSQLS', 'SL');
parkSQ
> SELECT btrim(encode('SSparkSQLS', 'utf-8'), encode('SL', 'utf-8'));
parkSQ

```

## UCASE

- **Function Syntax:**

```
UCASE(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns str with all characters converted to uppercase.
- Return type: string
- Example:

```
> SELECT ucase('SparkSQL');  
SPARKSQL
```

## UNBASE64

- Function Syntax:

```
UNBASE64(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts str from a base64 string to binary.
- Return type: binary
- Example:

```
> SELECT unbase64('U3BhcmsgU1FM');  
Spark SQL
```

## UNHEX

- Function Syntax:

```
UNHEX(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Converts a hexadecimal str to binary.
- Return type: binary
- Example:

```
> select unhex('74656E63656E74');
```

```
tencent
```

## UPPER

- Function Syntax:

```
UPPER(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns str with all characters converted to uppercase.
- Return type: string
- Example:

```
> SELECT upper('tencent');  
TENCENT
```

## UUID

- Function Syntax:

```
UUID()
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a 36-character UUID.
- Return type: string
- Example:

```
> SELECT uuid();  
46707d92-02f4-4817-8116-a4c3b23e6266
```

## XPATH

- Function Syntax:

```
XPATH(<xml> string, <xpath> string)
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Returns an array of strings that match the XPath expression within the xml node.
- **Return type:** array<string>
- **Example:**

```
> SELECT xpath ('<a><b>b1</b><b>b2</b><b>b3</b><c>c1</c><c>c2</c></a>', 'a/b/text() ');  
[ "b1", "b2", "b3" ]
```

## XPATH\_BOOLEAN

- **Function Syntax:**

```
XPATH_BOOLEAN(<xml> string, <xpath> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns true if the XPath expression evaluates to true or if a matching node is found.
- **Return type:** boolean
- **Example:**

```
> SELECT xpath_boolean ('<a><b>1</b></a>', 'a/b');  
true
```

## XPATH\_DOUBLE

- **Function Syntax:**

```
XPATH_DOUBLE(<xml> string, <xpath> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a double value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- **Return Type:** double
- **Example:**

```
> SELECT xpath_double ('<a><b>1</b><b>2</b></a>', 'sum(a/b) ');  
3.0
```



## XPATH\_NUMBER

- Function Syntax:

```
XPATH_NUMBER(<xml> string, <xpath> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a double value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- Return Type: double
- Example:

```
> SELECT xpath_number('<a><b>1</b><b>2</b></a>', 'sum(a/b)');  
3.0
```

## XPATH\_FLOAT

- Function Syntax:

```
XPATH_FLOAT(<xml> string, <xpath> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a float value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- Return Type: float
- Example:

```
> SELECT xpath_float('<a><b>1</b><b>2</b></a>', 'sum(a/b)');  
3.0
```

## XPATH\_INT

- Function Syntax:

```
XPATH_INT(<xml> string, <xpath> string)
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Returns an int value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- **Return Type:** integer
- **Example:**

```
> SELECT xpath_int ('<a><b>1</b><b>2</b></a>', 'sum(a/b)');  
3
```

## XPATH\_LONG

- **Function Syntax:**

```
XPATH_LONG(<xml> string, <xpath> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a bigint value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- **Return type:** bigint
- **Example:**

```
> SELECT xpath_long ('<a><b>1</b><b>2</b></a>', 'sum(a/b)');  
3
```

## XPATH\_SHORT

- **Function Syntax:**

```
XPATH_SHORT(<xml> string, <xpath> string)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns a short value. It returns zero if no match is found, or NaN if a match is found but the value is not a number.
- **Return type:** short
- **Example:**

```
> SELECT xpath_short ('<a><b>1</b><b>2</b></a>', 'sum(a/b)');  
3
```

## XPATH\_STRING

- Function Syntax:

```
XPATH_STRING(<xml> string, <xpath> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the text content of the first xml node that matches the XPath expression.
- Return type: string
- Example:

```
> SELECT xpath_string('<a><b>b</b><c>cc</c></a>', 'a/c');  
cc
```

## REGEXP\_EXTRACT

- Function Syntax:

```
REGEXP_EXTRACT(<str> string, <regexp> string[, <idx> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Extracts the first string in str that matches the regexp expression and corresponds to the regex group index idx.
- Return type: string
- Example:

```
> SELECT regexp_extract('100-200', '(\d+)-(\d+)', 1);  
100
```

## REGEXP\_EXTRACT\_ALL

- Function Syntax:

```
REGEXP_EXTRACT_ALL(<str> string, <regexp> string[, <idx> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Extracts all strings in str that match the regexp expression and correspond to the regex group index.

- Return type: array<string>
- Example:

```
> SELECT regexp_extract_all('100-200, 300-400', '(\d+)-(\d+)', 1);  
["100", "300"]
```

## REGEXP\_REPLACE

- Function Syntax:

```
REGEXP_REPLACE(<str> string, <regexp> string, <rep> string[,  
<position> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Replaces all substrings in str that match regexp with rep.
- Return type: string
- Example:

```
> SELECT regexp_replace('100-200', '(\d+)', 'num');  
num-num
```

## REGEXP\_LIKE

- Function Syntax:

```
REGEXP_LIKE(<str> string, <regexp> string)
```

- Supported engines: SparkSQL.
- Usage: Returns true if the string matches the regular expression; otherwise, returns false.
- Return type: boolean
- Example:

```
> SELECT regexp_like('%SystemDrive%\Users\John',  
'%SystemDrive%\Users.*');  
true
```

## REGEXP

- **Function Syntax:**

```
REGEXP(<str> string, <regexp> string)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns true if the string matches the regular expression; otherwise, returns false.
- **Return type:** boolean
- **Example:**

```
> SELECT regexp('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');  
true
```

## CONCAT

- **Function Syntax:**

```
CONCAT(<s1> string, <s2> string, ...)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Concatenates s1, s2, ...
- **Return type:** string
- **Example:**

```
> SELECT concat('Spark', 'SQL');  
SparkSQL
```

## STR\_TO\_MAP

- **Function Syntax:**

```
str_to_map(<text> string[, <pairDelim> string[, <keyValueDelim>  
string]])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Creates a map by splitting the text into key/value pairs using delimiters. For pairDelim, the default delimiter is ','. For keyValueDelim, the default delimiter is ':'. Both pairDelim and keyValueDelim are treated as regular expressions.

- Return type: map<string, string>
- Example:

```
> SELECT str_to_map('a:1,b:2,c:3', ',', ':');
{"a":"1","b":"2","c":"3"}
> SELECT str_to_map('a');
{"a":null}
```

## REVERSE

- Function Syntax:

```
REVERSE(<str> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns a reversed string.
- Return type: string
- Example:

```
> SELECT reverse('Spark SQL');
LQS krapS
```

## RLIKE

- Function Syntax:

```
RLIKE(<str> string, <regex> string)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if str matches regex; otherwise, returns false.
- Return type: boolean
- Example:

```
> SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

## FROM\_CSV

- **Function Syntax:**

```
FROM_CSV(<csvStr> string, <schema> string, <options> map<string, string>)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns a structured value with the given csvStr and schema.
- **Return type:** struct
- **Example:**

```
> SELECT from_csv('1, 0.8', 'a INT, b DOUBLE');  
{"a":1,"b":0.8}  
> SELECT from_csv('26/08/2015', 'time Timestamp',  
map('timestampFormat', 'dd/MM/yyyy'));  
{"time":2015-08-26 00:00:00}
```

## SCHEMA\_OF\_CSV

- **Function Syntax:**

```
SCHEMA_OF_CSV(<csvStr> string[, options map<string, string>])
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the schema of a csv string.
- **Return type:** string
- **Example:**

```
> SELECT schema_of_csv('1,abc');  
STRUCT<`_c0`: INT, `_c1`: STRING>
```

## TO\_CSV

- **Function Syntax:**

```
TO_CSV(<expr> struct[, <options> map<string, string>])
```

- **Supported engines:** SparkSQL.

- **Usage:** Returns a csv string with the given structured value.
- **Return type:** string
- **Example:**

```
> SELECT to_csv(named_struct('a', 1, 'b', 2));
1,2
> SELECT to_csv(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')), map('timestampFormat', 'dd/MM/yyyy'));
26/08/2015
```

## NGRAMS

- **Function Syntax:**

```
NGRAMS(<a> array<array<string>>, <N> integer, <K> integer, <pf> integer)
```

- **Supported engines:** Presto.
- **Usage:** Returns the top k N-grams from a set of tokenized sentences.
- **Return type:** array<struct<string,double>>.

## CONTEXT\_NGRAMS

- **Function Syntax:**

```
CONTEXT_NGRAMS((array <array <string>>, array <string>, int, int))
```

- **Supported engines:** Presto.
- **Usage:** Given a contextual N-gram, returns the top k contextual N-grams from a set of tokenized sentences.
- **Return type:** array<struct<string, double>>.



# Aggregate Functions

Last updated: 2026-05-21 14:34:32

## APPROX\_COUNT\_DISTINCT

- Function Syntax:

```
APPROX_COUNT_DISTINCT(<expr> any [, <relativeSD>  
integer|double|decimal])
```

- Supported engines: SparkSQL.
- Usage: Returns the cardinality estimated by HyperLogLog++. The parameter `relativeSD` defines the maximum allowed relative standard deviation.
- Return type: bigint
- Example:

```
SELECT approx_count_distinct(col1) FROM (VALUES (1), (1), (2), (2),  
(3)) tab(col1);  
3
```

## AVG

- Function Syntax:

```
AVG(<expr> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the average value calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT avg(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
2.0  
> SELECT avg(col) FROM (VALUES (1), (2), (NULL)) AS tab(col);  
1.5
```

## CORR

- **Function Syntax:**

```
CORR(<expr> integer|double|decimal, <expr> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the Pearson correlation coefficient between a set of number pairs.
- **Return Type:** double
- **Example:**

```
> SELECT corr(c1, c2) FROM (VALUES (3, 2), (3, 3), (6, 4)) as tab(c1, c2);  
0.8660254037844387
```

## COUNT

- **Function Syntax:**

```
COUNT(*)  
COUNT([DISTINCT] <col1> ANY, <col2> ANY, ...)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:**  
**COUNT(\*):** Returns the total number of rows retrieved, including those containing null values.  
**COUNT(<col1> ANY, <col2> ANY, ...):** Returns the number of rows where all provided expressions are non-null.  
**COUNT([DISTINCT] <col1> ANY, <col2> ANY, ...):** Returns the number of rows where the provided expressions are unique and non-null.
- **Return Type:** integer
- **Example:**

```
> SELECT count(*) FROM (VALUES (NULL), (5), (5), (20)) AS tab(col);  
4  
> SELECT count(col) FROM (VALUES (NULL), (5), (5), (20)) AS tab(col);  
3  
> SELECT count(DISTINCT col) (FROM VALUES (NULL), (5), (5), (10)) AS  
tab(col);
```

```
2
```

## COUNT\_IF

- Function Syntax:

```
COUNT_IF (<expr> ANY)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the number of rows where the expression evaluates to TRUE.
- Return Type: int
- Example:

```
> SELECT count_if(col % 2 = 0) FROM (VALUES (NULL), (0), (1), (2),  
(3)) AS tab(col);  
2  
  
> SELECT count_if(col IS NULL) FROM (VALUES (NULL), (0), (1), (2),  
(3)) AS tab(col);  
1
```

## COVER\_POP

- Function Syntax:

```
COVAR_POP (<expr1> integer|double|decimal, <expr2>  
integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the population covariance of a set of number pairs.
- Return Type: double
- Example:

```
> SELECT covar_pop(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS  
tab(c1, c2);  
0.6666666666666666
```

## COVER\_SAMP

- **Function Syntax:**

```
COVAR_SAMP(<expr1> integer|double|decimal, <expr2>
integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the sample covariance of a set of number pairs.
- **Return Type:** double
- **Example:**

```
> SELECT covar_samp(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS
tab(c1, c2);
1.0
```

## FIRST\_VALUE

- **Function Syntax:**

```
FIRST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the first value of expr for a set of rows. If isIgnoreNull is true, only non-null values are returned.
- **Return Type:** T
- **Example:**

```
> SELECT first_value(col) FROM (VALUES (10), (5), (20)) AS tab(col);
10
> SELECT first_value(col) FROM (VALUES (NULL), (5), (20)) AS tab(col);
NULL
> SELECT first_value(col, true) FROM (VALUES (NULL), (5), (20)) AS
tab(col);
5
```

## FIRST

- **Function Syntax:**

```
FIRST(<expr> T[, <isIgnoreNull> boolean])
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the first value of expr for a set of rows. If isIgnoreNull is true, only non-null values are returned.
- **Return Type:** T
- **Example:**

```
> SELECT first(col) FROM (VALUES (10), (5), (20)) AS tab(col);
10
> SELECT first(col) FROM (VALUES (NULL), (5), (20)) AS tab(col);
NULL
> SELECT first(col, true) FROM (VALUES (NULL), (5), (20)) AS tab(col);
5
```

## KURTOSIS

- **Function Syntax:**

```
KURTOSIS(<expr> integer|double|decimal)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the kurtosis value calculated based on the group's values.
- **Return Type:** double
- **Example:**

```
> SELECT kurtosis(col) FROM (VALUES (-10), (-20), (100), (1000)) AS
tab(col);
-0.7014368047529627
> SELECT kurtosis(col) FROM (VALUES (1), (10), (100), (10), (1)) as
tab(col);
0.19432323191699075s
```

## LAST\_VALUE

- **Function Syntax:**

```
LAST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the last value of expr for a set of rows. If isIgnoreNull is true, only non-null values are returned.
- **Return Type:** T
- **Example:**

```
> SELECT last_value(col) FROM (VALUES (10), (5), (20)) AS tab(col);
20
> SELECT last_value(col) FROM (VALUES (10), (5), (NULL)) AS tab(col);
NULL
> SELECT last_value(col, true) FROM (VALUES (10), (5), (NULL)) AS
tab(col);
5
```

## LAST

- **Function Syntax:**

```
LAST(<expr> T[, <isIgnoreNull> boolean])
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the last value of expr for a set of rows. If isIgnoreNull is true, only non-null values are returned.
- **Return Type:** T
- **Example:**

```
> SELECT last(col) FROM (VALUES (10), (5), (20)) AS tab(col);
20
> SELECT last(col) FROM (VALUES (10), (5), (NULL)) AS tab(col);
NULL
> SELECT last(col, true) FROM (VALUES (10), (5), (NULL)) AS tab(col);
5
```

## MEAN

- **Function Syntax:**

```
MEAN(<expr> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the average value calculated based on the group's values.
- **Return Type:** double
- **Example:**

```
> SELECT mean(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
2.0  
> SELECT mean(col) FROM (VALUES (1), (2), (NULL)) AS tab(col);  
1.5
```

## PERCENTILE

- **Function Syntax:**

```
PERCENTILE(<col> ANY, <percentage>  
integer|double|decimal|array<double> [, <frequency> integer])
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the exact percentile value for the numeric column col at a given percentage. The percentage value must be between 0.0 and 1.0. The frequency should be a positive integer.
- **Return Type:** double
- **Example:**

```
> SELECT percentile(col, 0.3) FROM (VALUES (0), (10)) AS tab(col);  
3.0  
> SELECT percentile(col, array(0.25, 0.75)) FROM (VALUES (0), (10)) AS  
tab(col);  
[2.5, 7.5]
```

## SKEWNESS

- **Function Syntax:**

```
SKEWNESS(<col> integer|double|decimal)
```

- Supported engines: SparkSQL.
- Usage: Returns the skewness value calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT skewness(col) FROM (VALUES (-10), (-20), (100), (1000)) AS  
tab(col);  
1.1135657469022011  
> SELECT skewness(col) FROM (VALUES (-1000), (-100), (10), (20)) AS  
tab(col);  
-1.1135657469022011
```

## PERCENTILE\_APPROX

- Function Syntax:

```
PERCENTILE_APPROX(<col> integer|double|decimal, <percentage>  
double|array<double>[, <accuracy> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the approximate percentile for the numeric column col. This value is the minimum value among the ordered col values (sorted from smallest to largest), such that no more than the specified percentage of col values are less than or equal to it. The percentage value must be between 0.0 and 1.0. The accuracy parameter (default: 10000) is a positive numeric literal that controls the approximation accuracy at the cost of memory. A higher accuracy value yields better precision, and  $1.0/\text{accuracy}$  is the relative error of the approximation. When percentage is an array, each value in the percentage array must be between 0.0 and 1.0. In this case, an array of approximate percentiles for column col is returned based on the given percentage array.
- Return type: integer | array<integer>
- Example:

```
> SELECT percentile_approx(col, array(0.5, 0.4, 0.1), 100) FROM  
(VALUES (0), (1), (2), (10)) AS tab(col);  
[1,1,0]
```



```
> SELECT percentile_approx(col, 0.5, 100) FROM (VALUES (0), (6), (7),
(9), (10)) AS tab(col);
7
```

## APPROX\_PERCENTILE

- Function Syntax:

```
APPROX_PERCENTILE(<col> integer|double|decimal, <percentage>
double|array<double>[, <accuracy> integer])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the approximate percentile for the numeric column col. This value is the minimum value among the ordered col values (sorted from smallest to largest), such that no more than the specified percentage of col values are less than or equal to it. The percentage value must be between 0.0 and 1.0. The accuracy parameter (default: 10000) is a positive numeric literal that controls the approximation accuracy at the cost of memory. A higher accuracy value yields better precision, and 1.0/accuracy is the relative error of the approximation. When percentage is an array, each value in the percentage array must be between 0.0 and 1.0. In this case, an array of approximate percentiles for column col is returned based on the given percentage array.
- Return type: integer | array<integer>
- Example:

```
> SELECT APPROX_PERCENTILE(col, array(0.5, 0.4, 0.1), 100) FROM
(VALUE (0), (1), (2), (10)) AS tab(col);
[1,1,0]
> SELECT APPROX_PERCENTILE(col, 0.5, 100) FROM (VALUES (0), (6), (7),
(9), (10)) AS tab(col);
7
```

## MAX

- Function Syntax:

```
MAX(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Returns the maximum value of col.
- **Return type:** Same as col.
- **Example:**

```
> SELECT max(col) FROM (VALUES (10), (50), (20)) AS tab(col);  
50
```

## MAX\_BY

- **Function Syntax:**

```
MAX_BY(<x> T, <y> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the x value associated with the maximum value of y.
- **Return Type:** T
- **Example:**

```
> SELECT max_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c',  
20))) AS tab(x, y);  
b
```

## MIN

- **Function Syntax:**

```
MIN(<col> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the minimum value of col.
- **Return type:** Same as col.
- **Example:**

```
> SELECT min(col) FROM (VALUES (10), (50), (20)) AS tab(col);  
10
```

## MIN\_BY

- **Function Syntax:**

```
MIN_BY(<x> T, <y> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the x value associated with the minimum value of y.
- **Return Type:** T
- **Example:**

```
> SELECT min_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c',  
20))) AS tab(x, y);  
a
```

## STD

- **Function Syntax:**

```
STD(<col> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.

## Usage Instructions

SparkSQL: Returns the sample standard deviation calculated based on the group's values.

Presto: Returns the population standard deviation calculated based on the group's values.

- **Return Type:** double
- **Example:**

```
> SELECT std(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
1.0
```

## STDDEV

- **Function Syntax:**

```
STDDEV(<col> integer|double|decimal)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the sample standard deviation calculated based on the group's values.

- Return Type: double
- Example:

```
> SELECT stddev(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
1.0
```

## STDDEV\_POP

- Function Syntax:

```
STDDEV_POP(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the population standard deviation calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT stddev_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
0.816496580927726
```

## STDDEV\_SAMP

- Function Syntax:

```
STDDEV_SAMP(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the sample standard deviation calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT stddev_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
1.0
```

## SUM

- Function Syntax:

```
SUM(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the sum calculated based on the group's values.
- Return type: Same as col.
- Example:

```
> SELECT sum(col) FROM (VALUES (5), (10), (15)) AS tab(col);
30
> SELECT sum(col) FROM (VALUES (NULL), (10), (15)) AS tab(col);
25
> SELECT sum(col) FROM (VALUES (NULL), (NULL)) AS tab(col);
NULL
```

## VARIANCE

- Function Syntax:

```
VARIANCE(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the sample variance calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT VARIANCE(col) FROM (VALUES (1), (2), (3)) AS tab(col);
1.0
```

## VAR\_POP

- Function Syntax:

```
VAR_POP(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the population variance calculated based on the group's values.
- Return Type: double

- Example:

```
> SELECT var_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
0.6666666666666666
```

## VAR\_SAMP

- Function Syntax:

```
VAR_SAMP(<col> integer|double|decimal)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the sample variance calculated based on the group's values.
- Return Type: double
- Example:

```
> SELECT var_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);  
1.0
```

## HISTOGRAM\_NUMERIC

- Function Syntax:

```
HISTOGRAM_NUMERIC(<col> integer, <nb> integer)
```

- Supported engines: Presto.
- Usage: Computes a histogram for the numeric column in a group using nb non-uniformly spaced bins. The output is an array of (x, y) coordinates with a size of nb, representing the center and height of each bin.
- Return Type: array<struct {'x','y'}>
- Example:

```
> SELECT histogram_numeric(col, 5) FROM (VALUES (0), (1), (2), (10))  
AS tab(col);  
[{"x":0,"y":1.0},{ "x":1,"y":1.0},{ "x":2,"y":1.0},{ "x":10,"y":1.0}]
```

## COLLECT\_LIST

- **Function Syntax:**

```
COLLECT_LIST(<col> T)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Collects and returns a list of non-unique elements.
- **Return type:** array<T>
- **Example:**

```
> SELECT collect_list(col) FROM (VALUES (1), (2), (1)) AS tab(col);  
[1,2,1]
```

## COLLECT\_SET

- **Function Syntax:**

```
COLLECT_SET(<col> T)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Collects and returns a set of unique elements.
- **Return type:** array<T>
- **Example:**

```
> SELECT collect_set(col) FROM (VALUES (1), (2), (1)) AS tab(col);  
[1,2]
```

## COUNT\_MIN\_SKETCH

- **Function Syntax:**

```
COUNT_MIN_SKETCH(<col> T, <eps> double, <confidence> double, <seed>  
integer)
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns a count-min sketch for the column with the specified esp, confidence, and seed. The result is binary and can be deserialized into a CountMinSketch before use.
- **Return type:** binary

- Example:

```
> SELECT hex(count_min_sketch(col, 0.5d, 0.5d, 1)) FROM (VALUES (1),
(2), (1)) AS tab(col);
00000001000000000000000030000000100000004000000005D8D6AB900000000000000
000000000000000002000000000000001000000000000000
```

## EVERY

- Function Syntax:

```
EVERY(<col> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if all values in the column are true.
- Return type: boolean
- Example:

```
> SELECT every(col) FROM (VALUES (true), (true), (true)) AS tab(col);
true
> SELECT every(col) FROM (VALUES (NULL), (true), (true)) AS tab(col);
true
> SELECT every(col) FROM (VALUES (true), (false), (true)) AS tab(col);
false
```

## BOOL\_AND

- Function Syntax:

```
BOOL_AND(<col> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if all values in the column are true.
- Return type: boolean
- Example:

```
> SELECT bool_and(col) FROM (VALUES (true), (true), (true)) AS
tab(col);
```



```
true
> SELECT bool_and(col) FROM (VALUES (NULL), (true), (true)) AS
tab(col);
true
> SELECT bool_and(col) FROM (VALUES (true), (false), (true)) AS
tab(col);
false
```

## AND

- Function Syntax:

```
<expr1> AND <expr2>
```

- Supported engines: SparkSQL, Presto.

## Usage Instructions

### Logical AND

- Return type: boolean
- Example:

```
> SELECT true and true;
true
> SELECT true and false;
false
> SELECT true and NULL;
NULL
> SELECT false and NULL;
false
```

## OR

- Function Syntax:

```
<expr1> OR <expr2>
```

- Supported engines: SparkSQL, Presto.
- Usage: Logical OR

- Return type: boolean
- Example:

```
> SELECT true or false;
true
> SELECT false or false;
false
> SELECT true or NULL;
true
> SELECT false or NULL;
NULL
```

## ANY

- Function Syntax:

```
ANY(<col> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if at least one value in the column is true.
- Return type: boolean
- Example:

```
> SELECT any(col) FROM (VALUES (true), (false), (false)) AS tab(col);
true
> SELECT any(col) FROM (VALUES (NULL), (true), (false)) AS tab(col);
true
> SELECT any(col) FROM (VALUES (false), (false), (NULL)) AS tab(col);
false
```

## SOME

- Function Syntax:

```
SOME(<col> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if at least one value in the column is true.

- Return type: boolean
- Example:

```
> SELECT some(col) FROM (VALUES (true), (false), (false)) AS tab(col);
true
> SELECT some(col) FROM (VALUES (NULL), (true), (false)) AS tab(col);
true
> SELECT some(col) FROM (VALUES (false), (false), (NULL)) AS tab(col);
false
```

## BOOL\_OR

- Function Syntax:

```
BOOL_OR(<col> boolean)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns true if at least one value in the column is true.
- Return type: boolean
- Example:

```
> SELECT BOOL_OR(col) FROM (VALUES (true), (false), (false)) AS
tab(col);
true
> SELECT BOOL_OR(col) FROM (VALUES (NULL), (true), (false)) AS
tab(col);
true
> SELECT BOOL_OR(col) FROM (VALUES (false), (false), (NULL)) AS
tab(col);
false
```

## BIT\_AND

- Function Syntax:

```
BIT_AND(<col> integer|bigint)
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Returns the bitwise AND of all non-null input values. If none exist, returns null.
- **Return type:** Same as col.
- **Example:**

```
> SELECT bit_and(col) FROM (VALUES (3), (5)) AS tab(col);  
1
```

## BIT\_OR

- **Function Syntax:**

```
BIT_OR(<col> integer|bigint)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the bitwise OR of all non-null input values. If none exist, returns null.
- **Return type:** Same as col.
- **Example:**

```
> SELECT bit_or(col) FROM (VALUES (3), (5)) AS tab(col);  
7
```

## BIT\_XOR

- **Function Syntax:**

```
BIT_XOR(<col> integer|bigint)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Returns the bitwise XOR of all non-null input values. If none exist, returns null.
- **Return type:** Same as col.
- **Example:**

```
> SELECT bit_xor(col) FROM (VALUES (3), (5)) AS tab(col);  
6
```

## ARG\_MIN

- **Function Syntax:**

```
ARG_MIN(<col1>, <col2> | expr(col2))
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the row with the smallest value in the specified column col1, based on the given calculation expression.
  - col1: A sortable column name. If it is a constant, an arbitrary value is returned. If multiple minimum values exist, one row is returned randomly.
- **Return type:** Same as the type of col2 or expr(col2).
- **Example:**

```
> SELECT arg_min(dt, uid) from values (1, 'm1'), (2, 't2'), (3, 'z3')
as tab(dt, uid);
m1
> SELECT arg_min(dt, upper(uid)) from values (1, 'm1', 1), (2, 't2',
1), (3, 'z3', 2) as tab(dt, uid, gid) group by gid;
M1
Z3
```

## ARG\_MAX

- **Function Syntax:**

```
ARG_MAX(<col1>, <col2> | expr(col2))
```

- **Supported engines:** SparkSQL.
- **Usage:** Returns the row with the largest value in the specified column col1, based on the given calculation expression.
  - col1: A sortable column name. If it is a constant, an arbitrary value is returned. If multiple maximum values exist, one row is returned randomly.
- **Return type:** Same as the type of col2 or expr(col2).
- **Example:**

```
> SELECT arg_max(dt, uid) from values (1, 'm1'), (2, 't2'), (3, 'z3')
as tab(dt, uid);
z3
```

```
> SELECT arg_max(dt, upper(uid)) from values (1, 'm1', 1), (2, 't2', 1), (3, 'z3', 2) as tab(dt, uid, gid) group by gid;
T2
Z3
```

## MAP\_UNION\_SUM

- **Function Syntax:**

```
MAP_UNION_SUM(map<k, v> input)
```

- **Supported engines:** SparkSQL.
- **Usage:** Specifies a column and sums all values in a map by key.
  - **Note 1:** null values are allowed in the aggregated data. If all values are null, the output is null.
  - **Note 2:** null values are allowed for a key's value. In this case, they are converted to the number 0.
- **Return type:** Same as the input type.
- **Example:**

```
> SELECT map_union_sum(ids) from values (map(1, 1)), (map(1,2)), (map(2,1)) as tab(ids);
{1:3,2:1}
> SELECT idx, map_union_sum(ids) from values (map(1, 1), 1), (map(1,2), 1), (map(2,1), 1) as tab(ids, idx) group by idx;
1 {1:3,2:1}
```

# Window Functions

Last updated: 2026-05-21 14:34:52

## row\_number

- Function Syntax:

```
row_number()
```

- Supported engines: SparkSQL, Presto.
- Usage: Assign a unique, consecutive number to each row.
- Return Type: int
- Example:

```
SELECT a, b, row_number() OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b);
A1  1  1
A1  1  2
A1  2  3
A2  3  1
```

## rank

- Function Syntax:

```
rank()
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the rank of a value within a set of values. If ranks are equal, gaps are left in the ranking sequence.
- Return Type: int
- Example:

```
SELECT a, b, rank() OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
```

```
A1  1  1
A1  2  3
A2  3  1
```

## dense\_rank

- Function Syntax:

```
dense_rank()
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the rank of a value within a set of values. Unlike the rank function, dense\_rank does not create gaps in the ranking sequence when ranks are equal.
- Return Type: int
- Example:

```
SELECT a, b, dense_rank() OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  1
A1  2  2
A2  3  1
```

## percent\_rank()

- Function Syntax:

```
percent_rank()
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the percentile rank of a value within a set of values. The return value is expressed as a decimal between 0 and 1.
- Return Type: double
- Example:

```
SELECT a, b, percent_rank() OVER (PARTITION BY a ORDER BY b) FROM
VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
```



```
A1  1  0.0
A1  1  0.0
A1  2  1.0
A2  3  0.0
```

## cume\_dist

- Function Syntax:

```
cume_dist()
```

- Supported engines: SparkSQL, Presto.
- Usage: Calculates the position of a value within a partition relative to all values.
- Return Type: double
- Example:

```
SELECT a, b, cume_dist() OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  0.6666666666666666
A1  1  0.6666666666666666
A1  2  1.0
A2  3  1.0
```

## first\_value

- Function Syntax:

```
first_value(col)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the value of the first row of a column within a partition.
- Return type: The data type of the col column.
- Example:

```
SELECT a, b, first_value(b) OVER (PARTITION BY a ORDER BY b) FROM
VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  1
```

```
A1  2  1
A2  3  3
```

## last\_value

- Function Syntax:

```
last_value(col)
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the value of the last row of a column within a partition.
- Return Type: int
- Example:

```
SELECT a, b, last_value(b) OVER (PARTITION BY a ORDER BY b) FROM
VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  1
A1  2  2
A2  3  3
```

## lag

- Function Syntax:

```
lag(col[, n [, default]])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the value from the row that is n rows before the current row within the window. The default value for n is 1, and the default value for default is null. If the value in the nth row is null, null is returned. If no such offset row exists (for example, when the offset is 1, the first row of the window has no preceding row), the default value is returned. The first parameter is the column name, the second parameter is the preceding nth row, and the third parameter is the default value.
- Return type: The data type of the col column.
- Example:

```
SELECT a, b, lag(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  NULL
A1  1  1
A1  2  1
A2  3  NULL
```

## lead

- Function Syntax:

```
lead(col[, n[, default]])
```

- Supported engines: SparkSQL, Presto.
- Usage: Returns the value from the row that is n rows after the current row within the window. The default value for n is 1, and the default value for default is null. If the value in the nth row is null, null is returned. If no such offset row exists (for example, when the offset is 1, the last row of the window has no following row), the default value is returned. The first parameter is the column name, the second parameter is the following nth row, and the third parameter is the default value.
- Return type: The data type of the col column.
- Example:

```
SELECT a, b, lead(b) OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  2
A1  2  NULL
A2  3  NULL
```

## nth\_value

- Function Syntax:

```
nth_value(col[, n])
```

- Supported engines: SparkSQL, Presto.

- **Usage:** Returns the value from the nth row from the start of the window. n starts from 1. If ignoreNulls=true, nulls are skipped when the nth row is located. Otherwise, every row is counted towards n. If no such nth row exists (for example, when n is 10 and the window size is less than 10), null is returned. The first parameter is the column name, and the second parameter is the nth row from the start.
- **Return type:** The data type of the col column.
- **Example:**

```
SELECT a, b, nth_value(b, 2) OVER (PARTITION BY a ORDER BY b) FROM
VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  1
A1  2  1
A2  3  NULL
```

## ntile

- **Function Syntax:**

```
ntile(n)
```

- **Supported engines:** SparkSQL, Presto.
- **Usage:** Divides the rows of a window partition into n buckets and returns the bucket number for each row, ranging from 1 to n.
- **Return Type:** int
- **Example:**

```
SELECT a, b, ntile(2) OVER (PARTITION BY a ORDER BY b) FROM VALUES
('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b)
A1  1  1
A1  1  1
A1  2  2
A2  3  1
```

## CLUSTER\_SAMPLE

- **Function Syntax:**

```
CLUSTER_SAMPLE(<int> N[, <int> M]) over (PARTITION BY col1 ORDER BY col2)
```

- **Supported engines:** SparkSQL.
- **Usage:** Samples within a window according to a specified ratio or quantity.
  - **N:** Required. An integer type. When only N is specified, it indicates that N rows of data are sampled. The sampling result is close to N rows, but is not guaranteed to be exactly N rows.
  - **M:** Optional. An integer type. When M is specified, it indicates that data is sampled at a quantity of  $M/N$  \* the total number of rows within the window. The sampling result is close to  $M/N$  \* the total number of rows.
- **Return type:** boolean. A value of true indicates that the row is sampled, and a value of false indicates that it is not sampled.
- **Example:**

```
> SELECT a, b, cluster_sample(2) OVER (PARTITION BY a ORDER BY b) FROM
VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1', 1) tab(a, b);
A1 2 true
A1 1 true
A2 3 true
A1 1 false

> SELECT a, b from (select a, b, cluster_sample(2) OVER (PARTITION BY
a ORDER BY b) as c FROM VALUES ('A1', 2), ('A1', 1), ('A2', 3), ('A1',
1) tab(a, b)) where c;
A1 2
A1 1
A2 3
```

# Other Functions

Last updated: 2026-05-21 15:00:04

## FIELD

- Function Syntax:

```
FIELD(<val> T, <val1> T, <val2> T, ...)
```

- Supported Engine: Presto
- Usage: Returns the index of val in the list val1, val2, ..., or returns 0 if not found.
- It supports all primitive types and uses str.equals(x) to compare parameters. If val is NULL, it returns 0.
- Return Type: integer
- Example:

```
select field('world', 'say', 'hello', 'world');  
3
```

## COALESCE

- Function Syntax:

```
COALESCE(<expr1> T, <expr2> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns the first non-null parameter if it exists; otherwise, returns null.
- Return Type: integer
- Example:

```
> SELECT coalesce(NULL, 1, NULL);  
1
```

## EXPLODE

- Function Syntax:

```
EXPLODE(<expr> array<T> | map<K, V>)
```

- **Supported Engine:** SparkSQL
- **Usage:** Splits the elements of an array-type expr into multiple rows, or splits a map-type expr into multiple rows and columns. Uses the default column name `col` for array elements, or `key` and `value` for map elements.
- **Return Type:** row(col T) | row(key K, value V)
- **Example:**

```
SELECT explode(array(10, 20));  
10  
20
```

## EXPLODE\_OUTER

- **Function Syntax:**

```
EXPLODE_OUTER(<expr> array<T> | map<K, V>)
```

- **Supported Engine:** SparkSQL
- **Usage:** Splits the elements of an array-type expr into multiple rows, or splits a map-type expr into multiple rows and columns. Uses the default column name `col` for array elements, or `key` and `value` for map elements.
- **Return Type:** row(col T) | row(key K, value V)
- **Example:**

```
SELECT explode_outer(array(10, 20));  
10  
20
```

## GREATEST

- **Function Syntax:**

```
GREATEST(<expr1> T, <expr2> T, ...>)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns the maximum value among all parameters, skipping null values.
- Return Type: T
- Example:

```
> SELECT greatest(10, 9, 2, 4, 3);  
10
```

## IF

- Function Syntax:

```
IF(<expr1> boolean, <expr2> T, <expr3> U)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns expr2 if expr1 evaluates to true; otherwise, returns expr3.
- Return Type: T|U
- Example:

```
> SELECT if(1 < 2, 'a', 'b');  
a
```

## INLINE

- Function Syntax:

```
INLINE(a array<struct<f1:T1, ..., fn:Tn>>)
```

- Supported Engine: SparkSQL
- Usage: Decomposes a structured array into a table. Uses column names col1, col2, and so on by default.
- Return Type: row(T1, ..., Tn)
- Example:



```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));  
1  a  
2  b
```

## INLINE\_OUTER

- Function Syntax:

```
INLINE_OUTER(a array<struct<f1:T1,...,fn:Tn>>)
```

- Supported Engine: SparkSQL
- Usage: Decomposes a structured array into a table. Uses column names such as col1, col2, and so on by default.
- Return Type: row(T1, ..., Tn)
- Example:

```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));  
1  a  
2  b
```

## IN

- Function Syntax:

```
<expr1> IN(<expr2> T, <expr3> T, ...)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns true if expr1 equals any exprn.
- Return Type: boolean
- Example:

```
> SELECT 1 in(1, 2, 3);  
true  
> SELECT 1 in(2, 3, 4);  
false
```

## ISNAN

- Function Syntax:

```
ISNAN(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns true if expr is NaN; otherwise, returns false.
- Return Type: boolean
- Example:

```
> SELECT isnan(cast('NaN' as double));  
true
```

## IFNULL

- Function Syntax:

```
IFNULL(<expr1> T, <expr2> U)
```

- Supported Engine: SparkSQL
- Usage: Returns expr2 if expr1 is null; otherwise, returns expr1.
- Return Type: T|U
- Example:

```
> SELECT ifnull(NULL, array('2'));  
["2"]
```

## ISNULL

- Function Syntax:

```
ISNULL(<expr> T)
```

- Supported Engines: SparkSQL, Presto.

- **Usage:** Returns true if expr is null; otherwise, returns false.
- **Return Type:** boolean
- **Example:**

```
> SELECT isnull(1);  
false
```

## ISNOTNULL

- **Function Syntax:**

```
ISNOTNULL(<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Returns true if expr is not null; otherwise, returns false.
- **Return Type:** boolean
- **Example:**

```
> SELECT isnotnull(1);  
true
```

## LEAST

- **Function Syntax:**

```
LEAST(<expr1> T, <expr2> T, ...)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Returns the minimum value among all parameters, skipping null values.
- **Return Type:** T
- **Example:**

```
> SELECT least(10, 9, 2, 4, 3);  
2
```

## NANVL

- Function Syntax:

```
NANVL(<expr1> T, <expr2> U)
```

- Supported Engines: SparkSQL, Presto.
- Usage: nanvl(expr1, expr2). Returns expr1 if expr1 is not NaN; otherwise, returns expr2.
- Return Type: T|U
- Example:

```
> SELECT nanvl(cast('NaN' as double), 123);  
123.0
```

## NULLIF

- Function Syntax:

```
NULLIF(<expr1> T, <expr2> U)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns null if expr1 equals expr2; otherwise, returns expr1.
- Return Type: T
- Example:

```
> SELECT nullif(2, 2);  
NULL
```

## NVL

- Function Syntax:

```
NVL(<expr1> T, <expr2> U)
```

- Supported Engines: SparkSQL, Presto.

- **Usage:** Returns `expr2` if `expr1` is null; otherwise, returns `expr1`.
- **Return Type:** T|U
- **Example:**

```
> SELECT nvl(NULL, array('2'));  
["2"]
```

## NVL2

- **Function Syntax:**

```
NVL2(<expr1> T1, <expr2> T2, <expr3> T3)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Returns `expr2` if `expr1` is not null; otherwise, returns `expr3`.
- **Return Type:** T2|T3
- **Example:**

```
> SELECT nvl2(NULL, 2, 1);  
1
```

## POSEXPLODE

- **Function Syntax:**

```
POSEXPLODE(<expr> array<T>|map<K, V>)
```

- **Supported Engine:** SparkSQL
- **Usage:** Splits the elements of an array-type `expr` into multiple rows, or splits a map-type `expr` into multiple rows and columns. Uses the column name ``pos`` to indicate position, the default column name ``col`` for array elements, or ``key`` and ``value`` for map elements.
- **Return Type:** `row(pos integer, col T) | row(row integer, key K, value V)`
- **Example:**

```
> SELECT posexplode(array(10,20));
```

```
0  10
1  20
```

## POSEXPLODE\_OUTER

- Function Syntax:

```
POSEXPLODE_OUTER(<expr> array<T>|map<K, V>)
```

- Supported Engine: SparkSQL
- Usage: Splits the elements of an array-type expr into multiple rows, or splits a map-type expr into multiple rows and columns. Uses the column name `pos` to indicate position, the default column name `col` for array elements, or `key` and `value` for map elements.
- Return Type: row(pos integer, col T) | row(row integer, key K, value V)
- Example:

```
> SELECT posexplode_outer(array(10,20));
0  10
1  20
```

## STACK

- Function Syntax:

```
STACK(<n> integer, <expr0> T0, ..., <expr1> T1)
```

- Supported Engine: SparkSQL
- Usage: The stack(n, expr1, ..., exprk) function splits expr1, ..., exprk into n rows. By default, it uses column names col0, col1, and so on.
- Return Type: row(col0 T0, ..., coln Tn)
- Example:

```
> SELECT stack(2, 1, 2, 3);
1  2
3  NULL
```

## ASSERT\_TRUE

- Function Syntax:

```
ASSERT_TRUE(<expr> boolean)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Throws an exception if expr is not true.
- Return Type: boolean
- Example:

```
> SELECT assert_true(0 < 1);  
NULL
```

## RAISE\_ERROR

- Function Syntax:

```
RAISE_ERROR(<error> string)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Throws an expr exception.
- Return Type: string
- Example:

```
> SELECT raise_error('custom error message');  
java.lang.RuntimeException  
custom error message
```

## SPARK\_PARTITION\_ID

- Function Syntax:

```
SPARK_PARTITION_ID()
```

- **Supported Engine:** SparkSQL
- **Usage:** Returns the current partition id.
- **Return Type:** integer
- **Example:**

```
> SELECT spark_partition_id();  
0
```

## INPUT\_FILE\_NAME

- **Function Syntax:**

```
INPUT_FILE_NAME()
```

- **Supported Engine:** SparkSQL
- **Usage:** Returns the name of the file being read. If unavailable, returns an empty string.
- **Return Type:** string
- **Example:**

```
> SELECT input_file_name();
```

## INPUT\_FILE\_BLOCK\_START

- **Function Syntax:**

```
INPUT_FILE_BLOCK_START()
```

- **Supported Engine:** SparkSQL
- **Usage:** Returns the starting offset of the block being read. If unavailable, returns -1.
- **Return Type:** integer
- **Example:**

```
> SELECT input_file_block_start();  
-1
```



## INPUT\_FILE\_BLOCK\_LENGTH

- Function Syntax:

```
INPUT_FILE_BLOCK_LENGTH()
```

- Supported Engine: SparkSQL
- Usage: Returns the length of the block being read. If unavailable, returns -1.
- Return Type: integer
- Example:

```
> SELECT input_file_block_length();  
-1
```

## MONOTONICALLY\_INCREASING\_ID

- Function Syntax:

```
MONOTONICALLY_INCREASING_ID()
```

- Supported Engine: SparkSQL
- Usage: Returns a monotonically increasing 64-bit integer. The generated ID is guaranteed to be monotonically increasing and unique, but not consecutive. In the current implementation, the partition ID occupies the high-order 31 bits, and the low-order 33 bits represent the record number within each partition. This assumes the DataFrame has fewer than 1 billion partitions, with each partition containing fewer than 8 billion records. This function is non-deterministic because its result depends on the partition ID.
- Return Type: bigint
- Example:

```
> SELECT monotonically_increasing_id();  
0
```

## CURRENT\_DATABASE

- Function Syntax:

```
CURRENT_DATABASE ()
```

- **Supported Engine:** SparkSQL
- **Usage:** Returns the current database.
- **Return Type:** string
- **Example:**

```
> SELECT current_database();  
default
```

## CURRENT\_CATALOG

- **Function Syntax:**

```
CURRENT_CATALOG ()
```

- **Supported Engine:** SparkSQL
- **Usage:** Returns the current catalog.
- **Return Type:** string
- **Example:**

```
> SELECT current_catalog();  
spark_catalog
```

## CURRENT\_USER

- **Function Syntax:**

```
CURRENT_USER ()
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Returns the current user.
- **Return Type:** string
- **Example:**

```
> SELECT current_user();
```

## REFLECT

- Function Syntax:

```
REFLECT(<class> string, <method> string[, <arg1> T1[, <arg2> T2,  
...]])
```

- Supported Engines: SparkSQL, Presto.
- Usage: Call a method that has reflection.
- Return Type: string
- Example:

```
> select reflect('java.lang.Math', 'abs', -1);  
1
```

## JAVA\_METHOD

- Function Syntax:

```
JAVA_METHOD(<class> string, <method> string[, <arg1> T1[, <arg2> T2,  
...]])
```

- Supported Engines: SparkSQL, Presto.
- Usage: Call a method that has reflection.
- Return Type: string
- Example:

```
> select JAVA_METHOD('java.lang.Math', 'abs', -1);  
1
```

## VERSION

- Function Syntax:

```
VERSION()
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns the engine version.
- Return Type: string
- Example:

```
> select VERSION()  
3.0.0 rce61711a5fa54ab34fc74d86d521ecaeea6b072a
```

## TYPEOF

- Function Syntax:

```
TYPEOF(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Returns the data type of expr.
- Return Type: string
- Example:

```
> SELECT typeof(1);  
int  
> SELECT typeof(array(1));  
array<int>
```

## CAST

- Function Syntax:

```
CAST(<expr> AS <type>)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Converts expr to the specified type.

- Return Type: <type>
- Example:

```
> SELECT cast('10' as int);  
10
```

## BOOLEAN

- Function Syntax:

```
BOOLEAN(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Converts expr to the boolean type.
- Return Type: boolean
- Example:

```
> SELECT boolean(1);  
true
```

## BIGINT

- Function Syntax:

```
BIGINT(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Performs a cast to the bigint type.
- Return Type: bigint
- Example:

```
> select bigint(0);  
0
```

## BINARY

- **Function Syntax:**

```
BINARY (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the BINARY type.
- **Return Type:** binary
- **Example:**

```
> select binary(65);  
A
```

## DOUBLE

- **Function Syntax:**

```
DOUBLE (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the double type.
- **Return Type:** double
- **Example:**

```
select double(1);  
1.0
```

## FLOAT

- **Function Syntax:**

```
FLOAT (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the float type.

- Return Type: float
- Example:

```
> select float(1);  
1.0
```

## INT

- Function Syntax:

```
INT(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Performs a cast to the integer type.
- Return Type: integer
- Example:

```
> select int(1.0);  
1
```

## SMALLINT

- Function Syntax:

```
SMALLINT(<expr> T)
```

- Supported Engines: SparkSQL, Presto.
- Usage: Performs a cast to the smallint type.
- Return Type: smallint
- Example:

```
select typeof(smallint(1));  
smallint
```

## STRING

- **Function Syntax:**

```
STRING (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the string type.
- **Return Type:** string
- **Example:**

```
> select typeof(string(1));  
string
```

## TINYINT

- **Function Syntax:**

```
TINYINT (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the tinyint type.
- **Return Type:** tinyint
- **Example:**

```
> select typeof(tinyint(1));  
tinyint
```

## DECIMAL

- **Function Syntax:**

```
DECIMAL (<expr> T)
```

- **Supported Engines:** SparkSQL, Presto.
- **Usage:** Performs a cast to the decimal type.
- **Return Type:** decimal



- Example:

```
> select typeof(decimal(1));  
decimal(10, 0)
```

## GET\_IDCARD\_BIRTHDAY

- Function Syntax:

```
GET_IDCARD_BIRTHDAY(<string> idcardno)
```

- Supported Engine: SparkSQL
- Usage: Obtains the date of birth from an ID card number.
  - idcardno: Required. It must be a string of either 15 or 18 digits representing a valid ID card number, and its validity is verified. Inputs of any other non-null type are not allowed. If the input is null, null is returned.
- Return Type: date
- Example (The following test ID card numbers are random samples and do not represent valid ID card numbers.)

```
> SELECT get_idcard_birthday('421081199001011222');  
1990-01-01
```

## GET\_IDCARD\_SEX

- Function Syntax:

```
GET_IDCARD_SEX(<string> idcardno)
```

- Supported Engine: SparkSQL
- Usage: Obtains the gender from an ID card number.
  - idcardno: Required. It must be a string of either 15 or 18 digits representing a valid ID card number, and its validity is verified. Inputs of any other non-null type are not allowed. If the input is null, null is returned.
- Return Type: string
- Example: (The following test ID card numbers are random samples and do not represent valid ID card numbers.)

```
> SELECT get_idcard_birthday('421081199001011222');
```

## GET\_IDCARD\_AGE

- Function Syntax:

```
GET_IDCARD_AGE(<string> idcardno)
```

- Supported Engine: SparkSQL
- Usage: Obtains the age from an ID card number.
  - idcardno: Required. It must be a string of either 15 or 18 digits representing a valid ID card number, and its validity is verified. Inputs of any other non-null type are not allowed. If the input is null, null is returned.
- Return Type: int
- Example: (The following test ID card numbers are random samples and do not represent valid ID card numbers.)

```
> SELECT get_idcard_age('421081197001021233');  
53
```

## MAX\_PT

- Function Syntax:

```
MAX_PT(<const string> tableFullName)
```

- Supported Engine: SparkSQL
- Usage: Obtains the maximum value from a specified partitioned table.
  - tableFullName: Required. It must be a string constant; otherwise, an error is reported. The tableFullName consists of three segments: catalog.db.table. If catalog and db are omitted, the configuration of the current session is used by default. It is recommended to specify the full name.
  - An error is reported if the input table is not a partitioned table.
  - This function returns only the largest partition **that contains data**. Because determining whether data exists involves the metadata center, **for external tables, you must check**

**whether the ANALYZE statement has been executed** to report the partition's statistics to the metadata.

- Partition values are sorted in lexicographic order to obtain the maximum value. When there are multiple partition levels, only the **first-level partition** is used for sorting.
- Return Type: string
- Example:

```
> SELECT max_pt('test.tableName');
20231024
> select * from test.tableName where dt=max_pt('test.tableName');
This is equivalent to `select * from test.tableName where
dt='20231014';`.
```

## TRANS\_ARRAY

- Function Syntax:

```
TRANS_ARRAY(<int> numKeys, <string> separator, <key1>, <key2>, ...,
<col1>, <col2>, ...)
```

- Supported Engine: SparkSQL
- Usage: Splits and transposes the specified columns (cols) into multiple rows. It also supports designating specific columns (keys) as the transpose key.
  - numKeys: Required. An integer that indicates the number of columns to be used as the transpose keys. The value must be greater than or equal to 0.
  - separator: Required. A string. When cols is a string, the string is split based on the separator. When cols is an array, you can enter any value for this parameter.
  - keys: Any type. The number of keys is determined by numKeys.
  - cols: A string or an array. All specified columns except for the keys are treated as cols, which are the columns to be split and transposed. **The data types of all cols must be the same**, meaning they must all be strings, all be arrays, or, as a special case, a combination of strings and string arrays. If no cols are specified, only one row is output.
  - Note 1: The total number of keys and cols must be greater than 0.
  - Note 2: If the split cols have unequal lengths, the number of rows after transposition is determined by the longest column. The other columns are padded with NULL.
- Return Type: Any type. The types of keys remain unchanged. The type of cols is the element type of a string or an array.

- Example:

```
> SELECT trans_array(1, ',', key, trans1, trans2) as (key, col1, col2)
from values ('1', '2,3', array('4', '5')) as tab (key, trans1,
trans2);
1 2 4
1 3 5

> SELECT trans_array(0, ',', key, trans1, trans2) as (key, col1, col2)
from values ('1', '2,3', array('4')) as tab (key, trans1, trans2);
1 2 4
NULL 3 NULL

> SELECT trans_array(3, ',', key, trans1, trans2) as (key, col1, col2)
from values ('1', '2,3', array('4', '5')) as tab (key, trans1,
trans2);
12,3 [4,5]
```

## TRANS\_COLS

- Function Syntax:

```
TRANS_COLS(<int> numKeys, <key1>, <key2>, ..., <col1>, <col2>, ...)
```

- Supported Engine: SparkSQL
- Usage: Transposes the specified columns (cols) into multiple rows. It also supports designating specific columns (keys) as the transpose key.
  - numKeys: Required. An integer that indicates the number of columns to be used as the transpose keys. The value must be greater than or equal to 0.
  - keys: Any type. The number of keys is determined by numKeys.
  - cols: Any type. All specified columns except for the keys are treated as cols, which are the columns to be transposed. **The data types of all cols must be the same.** If no cols are specified, only one row is output.
  - Note: The total number of keys and cols must be greater than 0.
- Return Type: Any type. The output includes a column named idx, which indicates the sequence number of the row among all transposed rows. The types of keys and cols remain unchanged.
- Example:

```
> SELECT trans_cols(1,sid,ip1,ip2) as (idx, sid, ip) from values
('s1','0.0.0.0','1.1.1.1') as tab(sid,ip1,ip2);
1 s1 0.0.0.0
2 s1 1.1.1.1
```

## SAMPLE

- Function Syntax:

```
SAMPLE(<int/long> totalParts[, <int/long> pointParts][, <col1>,
<col2>, ...])
```

- Supported Engine: SparkSQL
- Usage: This sampling function divides the data globally or based on the Hash values of specified columns into totalParts partitions, and returns the specified pointParts-th partition result.
  - totalParts: Required. An integer/long integer that indicates the total number of partitions to be created.
  - pointParts: Optional. An integer/long integer that indicates which partition of data to return. The default value is 1. The value must be less than or equal to totalParts.
  - cols: Optional. Any type. You can specify any number of columns as keys for data partitioning. The data is partitioned based on the hash values of these columns.
  - Note 1: If cols are not specified, or if all specified cols are NULL, random values are **used to replace hash values** for computation to prevent data skew. However, if cols are specified, the presence of at least one non-NULL column affects the hash value calculation and the partitioning result.
  - Note 2: Because the seeds for Hash and random values are fixed, the sampled data within a single partition remains consistent across repeated executions, provided the data order is unchanged. In scenarios involving small file merging, the sampling order may change. In such cases, it is recommended to specify columns for sampling.
- Return Type: boolean. A value of true indicates that the data is sampled, while a value of false indicates that it is not sampled.
- Example:

```
> select * from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, uid)
where sample(3, 1);
2 t2
```

```
→ select * from values (1, 'm1'), (2, 't2'), (3, 'z3') as tab(dt, uid)
where sample(3, 1, dt);
2 t2
3 z3
```

## TO\_CHAR

- Function Syntax:

```
TO_CHAR(<boolean|int|long|double|decimal> data [, <string> format])
TO_CHAR(<date|timestamp> data, string format)
```

- Supported Engine: SparkSQL
- Usage 1: Format numbers according to a specified template. The format parameter can be omitted, in which case numbers are directly converted to strings. Otherwise, the format parameter is defined as follows:
  - '0' or '9': Placeholders for digits. For the beginning of a string, if '0' is the placeholder, a missing digit is replaced with '0'; if '9' is the placeholder, a missing digit is replaced with a space. For the end of a string, any missing digit is uniformly replaced with '0'.
  - '.' or 'D': The placeholder for the decimal point. It can only appear once.
  - }
  - '\$': The dollar sign. It can only appear once.
  - 'S' or 'M': The placeholder for the minus sign and plus sign. It can appear at the beginning or end of a string at most once.
  - 'PR': It can only appear at the end of a string once. When the input data is a negative sign, the absolute value of the number is taken and enclosed in angle brackets.
- Usage 2: Format dates or times according to a specified template. The format parameter cannot be omitted and is defined as follows:
  - A template composed of any characters from yyyyMMdd HH:mm:ss.SSS.
  - For detailed reference, see: <https://spark.apache.org/docs/latest/sql-ref-datetime-pattern.html>
- Usage 3: TO\_CHAR supports implicit type conversion of its first argument. When the first argument is of the string type, it is implicitly converted to a timestamp, and the template logic for formatting dates or times is applied.
- Return Type: string.
- Example:

```
> select to_char(124.23);
124.23
> select to_char(124.23, '00999.9999');
<space><space>124.2300
> select to_char(4124.23, '9,999.99');
4,124.23
> select to_char(-124.23, '999.99S');
124.23-
> select to_char(-4124.23, '$9,999.99PR');
<$4,124.23>
> select to_char(date '2016-12-31 12:34:56', 'yyyymmddHHmmss');
20161231123456
```

# Presto Built-in Functions

Last updated: 2026-05-21 14:35:42

In addition to [Unified Functions](#), DLC also supports Presto built-in functions.

## Note:

The Presto engine is deprecated and only available for existing users.

## How to Enable Presto Built-in Functions

### Method 1: Configuring Functions for the Data Engine in Data Exploration

1. Log in to the [DLC console](#) and select a service region.
2. Go to Data Exploration and select a data engine. When the engine kernel is Presto, you can select the parameter **USEHIVEFUNCTION** in the advanced configuration. Set this parameter to **false** to use Presto built-in functions when you run SQL tasks with this data engine.

## Note:

In the current query session, Presto built-in functions can be used in all query tasks that use this data engine.

### Method 2: Adding Parameters to SQL Statements

To have an SQL task call Presto built-in functions, you can add configuration information to the SQL task.

Example:

```
SELECT /*+OPTIONS ('useHiveFunction'='false') */ prestofunc (xx)
```

### Method 3: Adding Configuration Parameters When Using APIs



In the config of the task structure, set the key-value pair `useHiveFunction=false`.

### SQLTask

SQL查询任务

被如下接口引用: `CreateTask`。

| 名称     | 类型                              | 必选 | 描述              |
|--------|---------------------------------|----|-----------------|
| SQL    | String                          | 是  | base64加密后的SQL语句 |
| Config | Array of <a href="#">KVPair</a> | 否  | 任务的配置信息         |

### Example:

```
Stringstatement="SELECTdate_add('week',3,TIMESTAMP'2001-08-2203:04:05.321')";
TasksInfotask=newTasksInfo();
task.setTaskType("SQLTask");
task.setSQL(Base64.getEncoder().encodeToString(statement.getBytes()));
// Add the following parameter configuration
KVPairpair=newKVPair();
pair.setKey("useHiveFunction");
pair.setValue("false");
task.setConfig(newKVPair[]{pair});

CreateTasksRequestrequest=newCreateTasksRequest();
request.setDatabaseName("");
request.setDataEngineName(PRESTO_ENGINE);
request.setTasks(task);
```

Refer to the API documentation: [Task Creation](#).

## Method 4: Adding Parameters When Tasks Are Created Using JDBC

Add the parameter `&presto.USEHIVEFUNCTION=true` to the JDBCURL path, or use `info.setProperty("presto.USEHIVEFUNCTION", "false");`.

```
Connectionconnection=
DriverManager.getConnection("jdbc:dlc:dlc.tencentcloudapi.com?
task_type=SQLTask&x...xx&presto.USEHIVEFUNCTION=true", info);
or
info.setProperty("presto.USEHIVEFUNCTION", "false");
```

```
info.setProperty("user","");
info.setProperty("password","");
```

# List of Supported Presto Built-in Functions

## Math Function

| Function Name           | Return Value | Function Description                                                                                                                                     |
|-------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| abs(x)                  | Same as x    | Returns the absolute value of x.                                                                                                                         |
| cbrt(x)                 | double       | Returning the cube root of x.                                                                                                                            |
| ceil(x)   ceiling(x)    | Same as x    | Returns the integer that is closest to and greater than or equal to x. For example: ceil(2.2); -->3.                                                     |
| cosine_similarity(x, y) | double       | Returns the similarity between vectors x and y. For example: SELECT cosine_similarity(MAP(ARRAY['a'], ARRAY[1.0]), MAP(ARRAY['a'], ARRAY[2.0])); -->1.0. |
| degrees(x)              | double       | Converting radians to degrees.                                                                                                                           |
| e()                     | double       | Returning the constant e.                                                                                                                                |
| exp(x)                  | double       | Returning e raised to the power of x.                                                                                                                    |
| floor(x)                | Same as x    | Returns the integer that is closest to and less than or equal to x. For example: floor(2.2); -->2.                                                       |
| ln(x)                   | double       | Returning the natural logarithm of x.                                                                                                                    |
| log2(x)                 | double       | Returning the base-2 logarithm of x.                                                                                                                     |
| log10(x)                | double       | Returning the base-10 logarithm of x.                                                                                                                    |
| mod(n, m)               | Same as n    | Returning the remainder of n divided by m.                                                                                                               |
| pi()                    | double       | Returning the constant Pi.                                                                                                                               |
| pow(x, p)   power(x, p) | double       | Returning x raised to the power of p.                                                                                                                    |
| radians(x)              | double       | Converting angle x (in degrees) to radians.                                                                                                              |
| rand()   random()       | double       | Returns a random value in the range 0.0 <= x <                                                                                                           |

|                                           |               |                                                                                                                                                                                                                                                                        |
|-------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                           |               | 1.0.                                                                                                                                                                                                                                                                   |
| random(n)                                 | Same as n     | Returns a random number between 0 and n (excluding n).                                                                                                                                                                                                                 |
| secure_rand()<br>  secure_random()<br>( ) | double        | Returns a cryptographically secure random value in the range $0.0 \leq x < 1.0$ .                                                                                                                                                                                      |
| secure_random(lower, upper)               | Same as lower | Returns a cryptographically secure random value in the range $\text{lower} \leq x < \text{upper}$ , where $\text{lower} < \text{upper}$ .                                                                                                                              |
| round(x)                                  | Same as x     | Returns x rounded to the nearest integer.                                                                                                                                                                                                                              |
| round(x, d)                               | Same as x     | Returns x rounded to d decimal places.                                                                                                                                                                                                                                 |
| sign(x)                                   | Same as x     | Returns the sign of x: returns 1 if $x > 0$ , returns -1 if $x < 0$ , and returns 0 if $x = 0$ .                                                                                                                                                                       |
| sqrt(x)                                   | double        | Returning the square root of x.                                                                                                                                                                                                                                        |
| truncate(x)                               | bigint        | Returns the number truncated after the decimal point.<br>For example: <code>truncate(4.9) --&gt; 4</code> .                                                                                                                                                            |
| truncate(x, n)                            | double        | Returns x truncated to n decimal places. When $n < 0$ , truncates from the left of the decimal point.<br>For example: <code>truncate(12.333, -1) --&gt; 10.0</code> ;<br><code>truncate(12.333, 0) --&gt; 12.0</code> ; <code>truncate(12.333, 1) --&gt; 12.3</code> . |

## Binary Functions

| Function Name       | Return Value | Function Description                                                                                                                                                                            |
|---------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| length(b)           | bigint       | Returns the binary length of b in bytes.                                                                                                                                                        |
| concat(b1, ..., bN) | varbinary    | Returns the binary value of concatenating b1, ..., bN.                                                                                                                                          |
| substr(b, start)    | varbinary    | Returns the binary value extracted from b starting at position start. When $\text{start} > 0$ , extraction starts from the beginning. When $\text{start} < 0$ , extraction starts from the end. |

|                                       |                        |                                                                                                                                                                                                                                                                              |
|---------------------------------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>substr(b, start, length)</code> | <code>varbinary</code> | Returns the binary value extracted from <code>b</code> starting at position <code>start</code> for a length of <code>length</code> . When <code>start &gt; 0</code> , extraction starts from the beginning. When <code>start &lt; 0</code> , extraction starts from the end. |
| <code>to_base64(b)</code>             | <code>varchar</code>   | Converts binary data <code>b</code> to a base64 string.                                                                                                                                                                                                                      |
| <code>from_base64(string)</code>      | <code>varbinary</code> | Converts a base64-encoded string to binary data.                                                                                                                                                                                                                             |
| <code>to_base64url(b)</code>          | <code>varchar</code>   | Converts binary data <code>b</code> to a base64 string using URL-safe characters.                                                                                                                                                                                            |
| <code>from_base64url(string)</code>   | <code>varbinary</code> | Converts a base64-encoded string to binary data using URL-safe characters.                                                                                                                                                                                                   |
| <code>from_base32(string)</code>      | <code>varbinary</code> | Converts a base32-encoded string to binary data.                                                                                                                                                                                                                             |
| <code>to_base32(b)</code>             | <code>varchar</code>   | Converts binary data <code>b</code> to a base32 string.                                                                                                                                                                                                                      |
| <code>to_hex(b)</code>                | <code>varchar</code>   | Converts binary data <code>b</code> to a hexadecimal string.                                                                                                                                                                                                                 |
| <code>from_hex(string)</code>         | <code>varbinary</code> | Converts a hexadecimal-encoded string to binary data.                                                                                                                                                                                                                        |
| <code>lpad(b, size, padb)</code>      | <code>varbinary</code> | Returns a binary number by left-padding <code>b</code> with <code>padb</code> until the length reaches <code>size</code> .                                                                                                                                                   |
| <code>rpadd(b, size, padb)</code>     | <code>varbinary</code> | Returns a binary number by right-padding <code>b</code> with <code>padb</code> until the length reaches <code>size</code> .                                                                                                                                                  |
| <code>crc32(b)</code>                 | <code>bigint</code>    | Computes the cyclic redundancy check value of the expression using the CRC32 algorithm.                                                                                                                                                                                      |
| <code>md5(b)</code>                   | <code>varbinary</code> | Computes the md5 value of binary data <code>b</code> .                                                                                                                                                                                                                       |
| <code>sha1(b)</code>                  | <code>varbinary</code> | Computes the sha1 value of binary data <code>b</code> .                                                                                                                                                                                                                      |
| <code>sha256(b)</code>                | <code>varbinary</code> | Computes the sha256 value of binary data <code>b</code> .                                                                                                                                                                                                                    |
| <code>sha512(b)</code>                | <code>varbinary</code> | Computes the sha512 value of binary data <code>b</code> .                                                                                                                                                                                                                    |
| <code>xxhash64(b)</code>              | <code>varbinary</code> | Computes the xxhash64 value of binary data <code>b</code> .                                                                                                                                                                                                                  |

## Bitwise Operation Functions

| Function Name                                  | Return Value                | Function Description                                                                                                              |
|------------------------------------------------|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <code>bit_count(x, bits)</code>                | bigint                      | Returning the number of bits in x.                                                                                                |
| <code>bitwise_and(x, y)</code>                 | bigint                      | Returning the bitwise AND of x and y.                                                                                             |
| <code>bitwise_not(x)</code>                    | bigint                      | Returning the bitwise NOT of x.                                                                                                   |
| <code>bitwise_or(x, y)</code>                  | bigint                      | Returning the bitwise OR of x and y.                                                                                              |
| <code>bitwise_xor(x, y)</code>                 | bigint                      | Returning the bitwise XOR of x and y.                                                                                             |
| <code>bitwise_left_shift(value, shift)</code>  | Same as the input parameter | Returning the value of value left-shifted by shift bits.<br>For example: <code>bitwise_left_shift(TINYINT '7', 2)</code> --> 28.  |
| <code>bitwise_right_shift(value, shift)</code> | Same as the input parameter | Returning the value of value right-shifted by shift bits.<br>For example: <code>bitwise_right_shift(TINYINT '7', 2)</code> --> 1. |

## String Functions

| Function Name                                       | Return Value | Function Description                                                                                                                                                 |
|-----------------------------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>chr(n)</code>                                 | varchar      | Returns the Unicode character for n.                                                                                                                                 |
| <code>codepoint(string)</code>                      | integer      | Returns the Unicode value of a string.<br>For example: <code>codepoint('0')</code> --> 48.                                                                           |
| <code>concat(string1, ..., stringN)</code>          | varchar      | Returns the concatenated string of string1, ..., stringN.                                                                                                            |
| <code>length(string)</code>                         | bigint       | Returns the length of a string.                                                                                                                                      |
| <code>hamming_distance(string1, string2)</code>     | bigint       | Returns the number of differing characters at corresponding positions in two strings of equal length.<br>For example: <code>hamming_distance(abc, art)</code> --> 2. |
| <code>levenshtein_distance(string1, string2)</code> | bigint       | Returns the minimum number of edit operations (insertion, deletion, substitution) required to transform one string into another.                                     |

|                                      |         |                                                                                                                                                                                                                                                                     |
|--------------------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                      |         | For example: levenshtein_distance('ab', 'abcde') --> 3.                                                                                                                                                                                                             |
| lower(string)                        | varchar | Returns the string converted to lowercase.                                                                                                                                                                                                                          |
| upper(string)                        | varchar | Returns the string converted to uppercase.                                                                                                                                                                                                                          |
| ltrim(string)                        | varchar | Removing leading whitespace from a string.                                                                                                                                                                                                                          |
| rtrim(string)                        | varchar | Returns the string with trailing whitespace removed.                                                                                                                                                                                                                |
| trim(string)                         | varchar | Returns the string with whitespace removed.                                                                                                                                                                                                                         |
| replace(string, search)              | varchar | Removing all occurrences of the search character from the string.                                                                                                                                                                                                   |
| replace(string, search, replace)     | varchar | Replacing the search string with the replace string.<br>For example: replace('abcavc', 'a', 'c') --> 'cbccvc'.                                                                                                                                                      |
| reverse(string)                      | varchar | Returning the characters in reverse order.<br>For example: reverse('abc') --> 'cba'.                                                                                                                                                                                |
| lpad(string, size, padstring)        | varchar | Left-pads the string with padstring until its length reaches size. If size is less than the length of string, string is truncated to length size.                                                                                                                   |
| rpadd(string, size, padstring)       | varchar | Right-pads the string with padstring until its length reaches size. If size is less than the length of string, string is truncated to length size.                                                                                                                  |
| split(string, delimiter)             | array   | Returns an array after the string is split by the delimiter.                                                                                                                                                                                                        |
| split(string, delimiter, limit)      | array   | Returns an array after the string is split by the delimiter and its size is limited to limit, where limit > 0. The last element of the array contains all remaining content from the string.<br>For example: select split('ab-cd-efg', '-', 2) --> ['ab','cd-efg']. |
| split_part(string, delimiter, index) | varchar | Returns the string at the index position in the array after the string is split by the delimiter. Field indexing starts from 1. If index > the array length, returns null.                                                                                          |

|                                                                                              |                                          |                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------------------------------------------------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                              |                                          | For example: <code>select split_part('ab-cd-efg', '-', 2) --&gt; 'cd'.</code>                                                                                                                                                                                                                                                                   |
| <code>split_to_map(string, entryDelimiter, keyValueDelimiter)</code>                         | <code>map&lt;varchar, varchar&gt;</code> | Returns a map after the string is split by <code>entryDelimiter</code> and <code>keyValueDelimiter</code> .<br>For example: <code>select split_to_map('a:1,b:2', ':', ':') --&gt; {'a':'1','b':'2'}.</code>                                                                                                                                     |
| <code>split_to_map(string, entryDelimiter, keyValueDelimiter, function(K, V1, V2, R))</code> | <code>map&lt;varchar, varchar&gt;</code> | Returns a map after the string is split by <code>entryDelimiter</code> and <code>keyValueDelimiter</code> . If duplicate keys exist, returns the value corresponding to the key according to the rule specified by function.<br>For example: <code>split_to_map('a:1,b:2;a:3', ':', ':', (k, v1, v2) -&gt; v1) --&gt; {'a':'1','b':'2'}.</code> |
| <code>substr(string, start)</code>                                                           | <code>varchar</code>                     | Returns the substring from position <code>start</code> in string <code>string</code> . Positions start from 1. When <code>start &lt; 0</code> , counting starts from the end of the string.                                                                                                                                                     |
| <code>substr(string, start, length)</code>                                                   | <code>varchar</code>                     | Returns a substring of length <code>length</code> starting from position <code>start</code> in string <code>string</code> . Positions start from 1. When <code>start &lt; 0</code> , counting starts from the end of the string.                                                                                                                |
| <code>position(substring IN string)</code>                                                   | <code>bigint</code>                      | Returns the position of the first occurrence of <code>substring</code> in <code>string</code> . Positions start from 1. If not found, returns 0.                                                                                                                                                                                                |
| <code>strpos(string, substring)</code>                                                       | <code>bigint</code>                      | Returns the position of the first occurrence of <code>substring</code> in <code>string</code> from left to right. Positions start from 1. If not found, returns 0.<br>For example: <code>strpos('abcdefg', 'a') --&gt; 1.</code>                                                                                                                |
| <code>strpos(string, substring, instance)</code>                                             | <code>bigint</code>                      | Returns the position of the <code>instance</code> -th occurrence of <code>substring</code> in <code>string</code> from left to right. <code>instance &gt; 0</code> . Positions start from 1. If not found, returns 0.<br>For example: <code>strpos('abcaefg', 'ab', 2) --&gt; 0.</code>                                                         |
| <code>strrpos(string, substring)</code>                                                      | <code>bigint</code>                      | Returns the position of the first occurrence of <code>substring</code> in <code>string</code> from right to left. Positions start from 1. If not found, returns 0.<br>For example: <code>strrpos('abcdefg', 'a') --&gt; 7.</code>                                                                                                               |
| <code>strrpos(string, substring,</code>                                                      | <code>bigint</code>                      | Returns the position of the <code>instance</code> -th occurrence of <code>substring</code> in <code>string</code> from right to                                                                                                                                                                                                                 |

|                            |           |                                                                                                                       |
|----------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------|
| instance)                  |           | left. instance > 0. Positions start from 1. If not found, returns 0.<br>For example: strpos('abcaefg', 'a', 2) --> 0. |
| to_utf8(string)            | varbinary | Converting a string to a UTF-8 encoded binary number.                                                                 |
| from_utf8(binary)          | varchar   | Converting a binary number to a UTF-8 encoded string, replacing invalid characters with the Unicode character U+FFFD. |
| from_utf8(binary, replace) | varchar   | Converting a binary number to a UTF-8 encoded string, replacing invalid characters with the specified replace string. |

## Date and Time Functions

| Function Name                    | Return Value | Function Description                                                                                                                                                               |
|----------------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| current_date                     | date         | Returns the current date.                                                                                                                                                          |
| current_time                     | time         | Returns the current time with time zone.                                                                                                                                           |
| current_timestamp                | timestamp    | Returns the current timestamp with time zone.                                                                                                                                      |
| current_timezone()               | varchar      | Returns the current time zone.                                                                                                                                                     |
| date(x)                          | date         | Returns the current date.                                                                                                                                                          |
| last_day_of_month(x)             | date         | Returns the last day of the month.                                                                                                                                                 |
| from_unixtime(unix time)         | timestamp    | Returns the unix timestamp.<br>For example: from_unixtime(1475996660) --> 2016-10-09 15:04:20.000                                                                                  |
| to_unixtime(timestamp)           | double       | Returns the unix timestamp of the timestamp.                                                                                                                                       |
| from_unixtime(unix time, string) | timestamp    | Returns the timestamp of unixtime, with the time zone specified by string.<br>For example:<br>from_unixtime(1617256617, 'Asia/Shanghai') --> 2021-04-01 13:56:57.000 Asia/Shanghai |
| from_iso8601_time                | timestamp    | Converts a string to a timestamp in ISO 8601                                                                                                                                       |



| stamp(string)                           |                             | format.                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| from_iso8601_date(string)               | date                        | Converts a string to a date in ISO 8601 format.                                                                                                                                                                                                                                                                                                                                        |
| localtime                               | time                        | Returns the current time.                                                                                                                                                                                                                                                                                                                                                              |
| localtimestamp                          | timestamp                   | Returns the current timestamp.                                                                                                                                                                                                                                                                                                                                                         |
| now()                                   | timestamp                   | Returns the current timestamp with time zone.                                                                                                                                                                                                                                                                                                                                          |
| to_iso8601(x)                           | varchar                     | Returns a string in ISO 8601 format for x, where x can be a date, timestamp, or timestamp with time zone.                                                                                                                                                                                                                                                                              |
| date_trunc(unit, x)                     | Same as the input parameter | Returns the time of x truncated to the unit, where unit can be second, minute, hour, day, week, month, quarter, or year.<br>For example: date_trunc('hour', TIMESTAMP '2020-03-17 02:09:30') --> '2020-03-17 02:00:00'.                                                                                                                                                                |
| date_add(unit, value, timestamp)        | Same as the input parameter | Returns a new timestamp calculated by adding or subtracting an interval of the specified unit, where a value > 0 indicates addition and a value < 0 indicates subtraction.<br>The unit can be millisecond, second, minute, hour, day, week, month, quarter, or year.<br>For example: date_add('hour', 2, cast ('2023-07-20 20:22:22.022' as timestamp)) --> '2023-07-20 22:22:22.022'. |
| date_diff(unit, timestamp1, timestamp2) | bigint                      | Returns the time difference between two timestamps in the specified unit. The unit can be millisecond, second, minute, hour, day, week, month, quarter, or year.<br>For example: date_diff('hour', cast ('2023-07-01 22:22:22' as timestamp), cast ('2023-07-03 22:22:22' as timestamp)) --> 48.                                                                                       |
| extract(field FROM x)                   | bigint                      | Returns the time component of x based on the specified field, where field can be: year, quarter, month, week, day, dow, doy, yow, hour, minute, second, timezone_hour, timezone_minute, day_of_week, day_of_year, or day_of_month.<br>For example: extract(year from current_date) --> 2023.                                                                                           |

|                 |        |                                                            |
|-----------------|--------|------------------------------------------------------------|
| day(x)          | bigint | Returns the day of the month for x.                        |
| day_of_month(x) | bigint | Returns the day of the month for x.                        |
| day_of_week(x)  | bigint | Returns the day of the week for x, ranging from 1 to 7.    |
| day_of_year(x)  | bigint | Returns the day of the year for x.                         |
| dow(x)          | bigint | Returns the day of the week for x. Same as day_of_week(x). |
| doy(x)          | bigint | Returns the day of the year for x. Same as day_of_year(x). |
| hour(x)         | bigint | Returns the hour of the day for x, ranging from 0–23.      |
| millisecond(x)  | bigint | Returns the millisecond of the second for x.               |
| minute(x)       | bigint | Returns the minute of the hour for x.                      |
| month(x)        | bigint | Returns the month of the year for x.                       |
| quarter(x)      | bigint | Returns the quarter of the year for x.                     |
| second(x)       | bigint | Returns the second of the minute for x.                    |
| week(x)         | bigint | Returns the week of the year for x.                        |
| week_of_year(x) | bigint | Returns the week of the year for x.                        |
| year(x)         | bigint | Returns the year of x.                                     |

Array Functions

| Function Name         | Return Value | Function Description                                          |
|-----------------------|--------------|---------------------------------------------------------------|
| array_distinct(x)     | array        | Removes duplicate values x from the array.                    |
| array_intersect(x, y) | array        | Returns the distinct elements in the intersection of x and y. |
| array_union(x, y)     | array        | Returns the distinct elements in the union of x and y.        |

|                                                         |          |                                                                                                                                                                                                    |
|---------------------------------------------------------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>array_except(x, y)</code>                         | array    | Returns the distinct elements that belong to x but not to y. (Difference set)                                                                                                                      |
| <code>array_join(x, delimiter, null_replacement)</code> | varchar  | Joins the elements of array x using the delimiter and fills null values in the array with null_replacement. null_replacement is an optional character.                                             |
| <code>array_max(x)</code>                               | x        | Returns the maximum value of the input array.                                                                                                                                                      |
| <code>array_min(x)</code>                               | x        | Returns the minimum value of the input array.                                                                                                                                                      |
| <code>array_position(x, element)</code>                 | bigint   | Returns the position (as a number) of the first occurrence of element in array x. If not found, returns 0.                                                                                         |
| <code>array_remove(x, element)</code>                   | array    | Removes all elements from array x that are identical to element.                                                                                                                                   |
| <code>array_sort(x)</code>                              | array    | Returns the sorted result of x. The elements of x must be sortable. Null elements are placed at the end of the returned array.                                                                     |
| <code>array_sort(array(T), function(T, T, int))</code>  | array(T) | Returns the array sorted according to the comparison function.<br>For example: <code>array_sort(ARRAY [3, 2, 5, 1, 2], (x, y) -&gt; IF(x &lt; y, 1, IF(x = y, 0, -1)))</code> --> [5, 3, 2, 2, 1]. |
| <code>arrays_overlap(x, y)</code>                       | Boolean  | Determines whether arrays x and y have any common non-null elements.                                                                                                                               |
| <code>cardinality(x)</code>                             | bigint   | Returns the cardinality (size) of an array.                                                                                                                                                        |
| <code>concat(array1,array 2,...,arrayN)</code>          | array    | Concatenates arrays array1, array2, ..., arrayN.                                                                                                                                                   |
| <code>contains(x, element)</code>                       | Boolean  | Determines whether array x contains element.                                                                                                                                                       |
| <code>element_at(array(E), index)</code>                | E        | Returns the element at the index position in the array. If index > 0, counts from left to right. If index < 0, counts from right to left.                                                          |
| <code>filter(array(T), funciton(T, boolean))</code>     | array(T) | Returns a new array constructed from the elements for which the function returns true.<br>For example: <code>filter( array[5, -6, NULL, 7], x -&gt; x &gt; 0);</code> --> [5, 7].                  |

|                                                  |               |                                                                                                                                                                                                                                 |
|--------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>repeat(element, count)</code>              | array         | Repeats the element count times.                                                                                                                                                                                                |
| <code>reverse(x)</code>                          | array         | Returns an array x with the reverse order.                                                                                                                                                                                      |
| <code>sequence(start, stop, n)</code>            | array(bigint) | Generates an integer sequence from start to stop with a step size of n, where n is optional and defaults to 1 if not filled in. If start is less than or equal to stop, increments by 1 each time; otherwise, increments by -1. |
| <code>shuffle(x)</code>                          | array         | Shuffles array elements.                                                                                                                                                                                                        |
| <code>slice(x, start, length)</code>             | array         | Returns an array sliced from the x array starting at position start with a length of length.                                                                                                                                    |
| <code>transform(array(T), function(T, U))</code> | array(U)      | Returns an array composed of element U, which is generated by applying the function to each element T in the array.<br>For example: <code>transform(array [5, 6], x -&gt; x + 1)</code> ; -- [6, 7].                            |

## JSON Functions

| Function Name                                 | Return Value | Function Description                                                                                                                                                                                                                          |
|-----------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>is_json_scalar(json)</code>             | boolean      | Determines whether the json is a json number, a json string, true, false, or null.<br>For example: <code>is_json_scalar('[1, 2, 3]')</code> --> false.                                                                                        |
| <code>json_array_contains(json, value)</code> | boolean      | Determines whether the value exists in the json (a string containing a json array).<br>For example: <code>json_array_contains('[1, 2, 3]', 2)</code> --> true.                                                                                |
| <code>json_array_length(json)</code>          | bigint       | Returns the array length of the json (a string containing a json array).<br>For example: <code>json_array_length('[1, 2, 3]')</code> --> 3.                                                                                                   |
| <code>json_extract(json, json_path)</code>    | json         | Evaluates a JSONPath-like expression, json_path, on the json (a string containing json) and returns the result as json.<br>For example: <code>json_extract('{ "log": { "file": { "path": "/etc/nginx/logs/access.log" }, "offset": 199</code> |

|                                      |         |                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                      |         | 91212}}', '\$.log.file.path') --> "/etc/nginx/logs/access.log".                                                                                                                                                                                                                                                                         |
| json_extract_scalar(json, json_path) | varchar | Similar to json_extract(), but returns the result value as a string, not a json string. The value referenced by json_path must be a boolean, a number, or a string.<br>For example: json_extract_scalar('{"log":{"file":{"path":"/etc/nginx/logs/access.log"},"offset":19991212}}', '\$.log.file.path') --> /etc/nginx/logs/access.log. |
| json_format(json)                    | varchar | Returns the json text serialized from the input json value. This is the inverse function of json_parse().<br>For example: json_format(JSON '[1, 2, 3]') --> [1,2,3].                                                                                                                                                                    |
| json_parse(string)                   | json    | Returns the json value deserialized from the input json text. This is the inverse function of json_format().<br>For example: json_parse('[1, 2, 3]') --> [1,2,3].                                                                                                                                                                       |
| json_size(json, json_path)           | bigint  | Similar to json_extract(), but returns the size. For an object or an array, the size is the number of members; for a scalar value, the size is zero. For example: json_size('{"x": [1, 2, 3]}', '\$.x') --> 3.                                                                                                                          |

## Aggregation Functions

| Function Name                         | Return Value                | Function Description                                                 |
|---------------------------------------|-----------------------------|----------------------------------------------------------------------|
| arbitrary(x)                          | Same as the input parameter | Returns any non-null value of x if it exists.                        |
| array_agg(x)                          | array[x]                    | Returns an array created from the input elements of x.               |
| avg(x)                                | double                      | Returns the average (arithmetic mean) of all input values.           |
| bool_and(boolean)<br>  every(boolean) | boolean                     | Returns true if every input value is true; otherwise, returns false. |

|                                |                               |                                                                                                                      |
|--------------------------------|-------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <code>bool_or(boolean)</code>  | boolean                       | Returns true if at least one input value is true; otherwise, returns false.                                          |
| <code>checksum(x)</code>       | varbinary                     | Returns the checksum of the input value.                                                                             |
| <code>count(*)</code>          | bigint                        | Number of Rows Returned                                                                                              |
| <code>count(x)</code>          | bigint                        | Returns the number of non-null input values.                                                                         |
| <code>count_if(x)</code>       | bigint                        | Returns the count of input values that are true, which is equivalent to <code>count(CASE WHEN x THEN 1 END)</code> . |
| <code>geometric_mean(x)</code> | double                        | Returns the geometric mean.                                                                                          |
| <code>max_by(x, y)</code>      | Same as the input parameter x | Returns the value from x that is associated with the maximum value of y.                                             |
| <code>max_by(x, y, n)</code>   | array[x]                      | Returns the n values from x that are associated with the n smallest values of y, sorted in descending order of y.    |
| <code>min_by(x, y)</code>      | Same as the input parameter x | Returns the value from x that is associated with the minimum value of y.                                             |
| <code>min_by(x, y, n)</code>   | array[x]                      | Returns the n values from x that are associated with the n smallest values of y, sorted in ascending order of y.     |
| <code>max(x)</code>            | Same as the input parameter   | Returns the maximum value in the input.                                                                              |
| <code>max(x, n)</code>         | Same as the input parameter x | Returns the n largest values in the input.                                                                           |
| <code>min(x)</code>            | Same as the input parameter   | Returns the minimum value in the input.                                                                              |

|                                       |                                            |                                                                                                                                                                                                                       |
|---------------------------------------|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>min(x, n)</code>                | Same as the input parameter <code>x</code> | Returns the <code>n</code> smallest values in the input.                                                                                                                                                              |
| <code>set_agg(x)</code>               | Same as the input parameter                | Returns an array of distinct elements.                                                                                                                                                                                |
| <code>set_union(array(T))</code>      | <code>array(T)</code>                      | Returns an array containing all distinct values from each input array.<br>For example: <code>select set_union(elements) FROM (VALUES ARRAY[1, 2, 3], ARRAY[2, 3, -4]) AS t(elements) --&gt; ARRAY[1, 2, 3, -4]</code> |
| <code>sum(x)</code>                   | Same as the input parameter                | Returns the sum.                                                                                                                                                                                                      |
| <code>bitwise_and_agg(x)</code>       | <code>bigint</code>                        | Returns the bitwise AND of all bits.                                                                                                                                                                                  |
| <code>bitwise_or_agg(x)</code>        | <code>bigint</code>                        | Returns the bitwise OR of all bits.                                                                                                                                                                                   |
| <code>histogram(x)</code>             |                                            | Returns a map containing the frequency of each distinct input value.                                                                                                                                                  |
| <code>map_agg(key, value)</code>      | <code>map(K, V)</code>                     | Returns a map composed of key-value pairs.                                                                                                                                                                            |
| <code>map_union(x(K, V))</code>       | <code>map(K, V)</code>                     | Returns the union of all input maps. If a key corresponds to multiple values, the value for that key in the result set is any one of those values.                                                                    |
| <code>map_union_sum(x(K, V))</code>   | <code>map(K, V)</code>                     | Returns a union map with values summed for identical keys. If the value corresponding to a key is null, it is counted as 0.                                                                                           |
| <code>multimap_agg(key, value)</code> |                                            | Returns a map composed of key-value pairs, where a key can correspond to multiple values.                                                                                                                             |

## Window Functions

| Function Name | Return Value | Function Description |
|---------------|--------------|----------------------|
|---------------|--------------|----------------------|

|                                            |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>row_number()</code>                  | <code>bigint</code>         | Assigns a unique, consecutive number to each row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>rank()</code>                        | <code>bigint</code>         | Calculate the ranking of a certain value in a group of values. If equal rankings occur, gaps are left in the ranking sequence.                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>percent_rank()</code>                | <code>double</code>         | Calculate the percentage rank of a value within a set of values. The return value is represented as a decimal ranging from 0 to 1.                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>dense_rank()</code>                  | <code>bigint</code>         | Calculate the ranking of a certain value in a group of values. If equal rankings occur, unlike the rank function, <code>dense_rank</code> does not leave gaps in the ranking sequence.                                                                                                                                                                                                                                                                                                                                     |
| <code>cume_dist()</code>                   | <code>bigint</code>         | Calculate the position of a value relative to all values in a partition.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>ntile(n)</code>                      | <code>bigint</code>         | Divides the rows of a window partition into <code>n</code> buckets and returns the bucket number for each row, ranging from 1 to <code>n</code> .                                                                                                                                                                                                                                                                                                                                                                          |
| <code>first_value(x)</code>                | Same as the input parameter | Returns the value of the first row for a specified column within a partition.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>last_value(x)</code>                 | Same as the input parameter | Returns the value of the last row for a specified column within a partition.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>nth_value(x, n)</code>               | Same as the input parameter | Returns the value of the <code>n</code> th row from the start of the window. <code>n</code> starts from 1.<br>If <code>ignoreNulls=true</code> , nulls are skipped when searching for the <code>n</code> th row. Otherwise, every row is counted towards <code>n</code> . If no such <code>n</code> th row exists (for example, when <code>n</code> is 10 and the window size is less than 10), null is returned. The first parameter is the column name, and the second parameter is the <code>n</code> th preceding row. |
| <code>lead(x[, n[, default_value]])</code> | Same as the input           | Returns the value of the <code>n</code> th row following the current row within the window. The default value for <code>n</code> is 1, and the default value for <code>default</code> is null.<br>If the value of the <code>n</code> th row is null, null is returned.                                                                                                                                                                                                                                                     |



|                                           |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                           | parameter                   | If no such offset row exists (for example, when the offset is 1 and the last row of the window has no following row), the default value is returned. The first parameter is the column name, the second parameter is the nth preceding row, and the third parameter is the default value.                                                                                                                                                                                                                      |
| <code>lag(x[, n[, default_value]])</code> | Same as the input parameter | Returns the value of the nth row preceding the current row within the window. The default value for n is 1, and the default value for default is null.<br>If the value of the nth row is null, null is returned.<br>If no such offset row exists (for example, when the offset is 1 and the first row of the window has no preceding row), the default value is returned. The first parameter is the column name, the second parameter is the nth preceding row, and the third parameter is the default value. |

## URL Functions

| Function Name                                 | Return Value | Function Description                                                                                |
|-----------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------|
| <code>url_extract_host(url)</code>            | varchar      | Returns the host of the URL                                                                         |
| <code>url_extract_parameter(url, name)</code> | varchar      | Returns the first value of the parameter 'name' in the url, as specified in RFC 1866#section-8.2.1. |
| <code>url_extract_path(url)</code>            | varchar      | Returns the path of the url.                                                                        |
| <code>url_extract_port(url)</code>            | bigint       | Returns the port of the url.                                                                        |
| <code>url_extract_protocol(url)</code>        | varchar      | Returns the protocol of the url.                                                                    |
| <code>url_extract_query(url)</code>           | varchar      | Returns the query string of the url.                                                                |
| <code>url_encode(value)</code>                | varchar      | Encoding the value                                                                                  |
| <code>url_decode(value)</code>                | varchar      | Decoding the url encoded by encode                                                                  |

## Other Functions

| Function Name                                | Return Value         | Function Description                                                                                                                                                                                      |
|----------------------------------------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>uuid ( )</code>                        | <code>uuid</code>    | Returns a randomly generated UUID.                                                                                                                                                                        |
| <code>cast(value AS type)</code>             | <code>type</code>    | Casts value to type.                                                                                                                                                                                      |
| <code>try_cast(value AS type)</code>         | <code>type</code>    | Casts value to type, and returns null if the cast fails.                                                                                                                                                  |
| <code>typeof(expr)</code>                    | <code>varchar</code> | Returns the type name of expr.<br>For example: <code>typeof(cos(2) + 1.5) --&gt; double</code> .                                                                                                          |
| <code>regexp_extract(string, pattern)</code> | <code>varchar</code> | Returns the first substring in string that matches the regular expression pattern.<br>For example: <code>regexp_extract('1a 2b 14m', '\d+') -&gt; 1</code> .                                              |
| <code>regexp_replace(string, pattern)</code> | <code>varchar</code> | Removes substrings matched by the regular expression pattern from string.<br>For example: <code>regexp_replace('1a 2b 14m', '\d+[ab] ') --&gt; '14m'</code> .                                             |
| <code>regexp_split(string, pattern)</code>   |                      | Splits the string using the regular expression pattern and returns an array, preserving trailing empty strings.<br>For example: <code>regexp_split('1a 2b 14m', '\s*[a-z]+\s*'); -- [1, 2, 14, ]</code> . |
| <code>regexp_like(string, pattern)</code>    | <code>boolean</code> | Determines whether there is a substring in string that matches the regular expression pattern.                                                                                                            |

# Hive Function Reference

Last updated: 2026-05-21 14:36:17

DLC provides comprehensive support for Hive functions. You can easily upgrade from Hive to DLC and leverage more powerful data lake features.

DLC unified functions have some minor differences from Hive functions. The detailed function comparison table is as follows.

For details about Hive functions, refer to the [LanguageManual UDF](#) on the Apache Hive website.

## Math Function

For details about Hive mathematical functions, refer to the [MathematicalFunctions](#) page on the Apache Hive website.

For DLC mathematical functions, see the [Mathematical Functions](#) document on the Tencent Cloud website.

| Hive 3.1 Function Name | Function Description                             | DLC Function Name | Difference Description                                                                                                                   | Usage Reference         |
|------------------------|--------------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| round                  | Rounding to the nearest integer                  | round             | No difference                                                                                                                            | select round(1.23);     |
| round                  | Rounding to a specified number of decimal places | round             | No difference                                                                                                                            | select round(1.234,2);  |
| bround                 | Rounding in HALF_EVEN mode                       | bround            | The bround function in DLC requires two parameters, with the second parameter specifying the number of decimal places to retain. Setting | select bround(1.237,0); |

|                            |                                                                                                                      |                            |                                                                                            |                                                                       |
|----------------------------|----------------------------------------------------------------------------------------------------------------------|----------------------------|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
|                            |                                                                                                                      |                            | the second parameter to 0 yields the same effect as <code>bround(double a)</code> in hive. |                                                                       |
| <code>bround</code>        | Rounding in HALF_EVEN mode to a specified number of decimal places                                                   | <code>bround</code>        | No difference                                                                              | <code>select bround(1.237,2);</code>                                  |
| <code>floor</code>         | Floor                                                                                                                | <code>floor</code>         | No difference                                                                              | <code>select floor(1.23);</code>                                      |
| <code>ceil, ceiling</code> | Ceil                                                                                                                 | <code>ceil, ceiling</code> | No difference                                                                              | <code>select ceil(1.93);</code><br><code>select ceiling(1.13);</code> |
| <code>rand</code>          | Returns a random number in the range of 0 to 1. If a seed is specified, a stable random number sequence is obtained. | <code>rand</code>          | No difference                                                                              | <code>select rand();</code><br><code>select rand(5);</code>           |
| <code>exp</code>           | e raised to the power of a                                                                                           | <code>exp</code>           | No difference                                                                              | <code>select exp(1);</code><br><code>select exp(2.4);</code>          |
| <code>ln</code>            | Natural logarithm function                                                                                           | <code>ln</code>            | No difference                                                                              | <code>select ln(2.4);</code>                                          |
| <code>log10</code>         | Base-10 logarithm function                                                                                           | <code>log10</code>         | No difference                                                                              | <code>select log10(2.4);</code>                                       |
| <code>log2</code>          | Base-2 logarithm function                                                                                            | <code>log2</code>          | No difference                                                                              | <code>select log2(2.4);</code>                                        |

|               |                                                            |             |                                              |                                        |
|---------------|------------------------------------------------------------|-------------|----------------------------------------------|----------------------------------------|
| log           | Logarithm function                                         | log         | No difference                                | select log(2,4);                       |
| pow,<br>power | Power function                                             | pow , power | No difference                                | select pow(2,4);                       |
| sqrt          | Square root function                                       | sqrt        | No difference                                | select sqrt(4);                        |
| bin           | Binary function                                            | bin         | No difference                                | select bin(4);                         |
| hex           | Hexadecimal function                                       | hex         | No difference                                | select hex(10);                        |
| unhex         | Converting hexadecimal to string                           | unhex       | decode the result in the DLC Spark function. | select decode(unhex('3141'), 'UTF-8'); |
| conv          | Base conversion                                            | conv        | No difference                                | select conv(2,10,2);                   |
| abs           | Absolute value                                             | abs         | No difference                                | select abs(-1);                        |
| pmod          | Modulo                                                     | pmod        | No difference                                | select pmod(5,3);                      |
| sin           | Trigonometric function sin, with a in radians.             | sin         | No difference                                | select sin(3.14);                      |
| asin          | Inverse trigonometric function arc sin, with a in radians. | asin        | No difference                                | select asin(0.5);                      |
| cos           | Trigonometric function cos, with a in radians.             | cos         | No difference                                | select cos(3.14);                      |
| acos          | Inverse trigonometric function arc cos, with a in radians. | acos        | No difference                                | select acos(1);                        |

|            |                                                                                         |            |               |                         |
|------------|-----------------------------------------------------------------------------------------|------------|---------------|-------------------------|
| tan        | Trigonometric function tan, with a in radians.                                          | tan        | No difference | select tan(3.14);       |
| atan       | Inverse trigonometric function arc tan, with a in radians.                              | atan       | No difference | select atan(1);         |
| degrees    | Converting radians to degrees                                                           | degrees    | No difference | select degrees(3.14);   |
| radians    | Converting degrees to radians                                                           | radians    | No difference | select radians(180);    |
| positive   | Returns a                                                                               | positive   | No difference | select positive(-1);    |
| negative   | Returns -a                                                                              | negative   | No difference | select negative(1);     |
| sign       | Returns 1.0 if a is positive, and returns -1.0 if a is negative. Returns 0.0 if a is 0. | sign       | No difference | select sign(1.12);      |
| e          | Returns the mathematical constant e                                                     | e          | No difference | select e();             |
| pi         | Returns the mathematical constant pi                                                    | pi         | No difference | select pi();            |
| factorial  | Factorial                                                                               | factorial  | No difference | select factorial(5);    |
| cbrt       | Cube root                                                                               | cbrt       | No difference | select cbrt(27);        |
| shiftright | Left shift                                                                              | shiftright | No difference | select shiftright(3,1); |
| shiftright | Right shift                                                                             | shiftright | No difference | select shiftright(3,1); |

|                     |                                                                                                        |                     |                                                                                                                                                                                           |                                 |
|---------------------|--------------------------------------------------------------------------------------------------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| shiftright unsigned | Unsigned right shift                                                                                   | shiftright unsigned | No difference                                                                                                                                                                             | select shiftrightunsigned(3,1); |
| greatest            | Returns the maximum value.                                                                             | greatest            | When a parameter contains a null value, the DLC function excludes null values from calculation and returns the maximum value among non-null values, while the Hive function returns null. | select greatest(1,2,3.3);       |
| least               | Returns the minimum value.                                                                             | least               | When a parameter contains a null value, the DLC function excludes null values from calculation and returns the minimum value among non-null values, while the Hive function returns null. | select least(1,2,3.3);          |
| width_bucket        | Creates num_buckets+1 equally sized buckets based on min_value/max_value and returns the bucket number | width_bucket        | No difference                                                                                                                                                                             | select width_bucket(10,1,20,2); |

where the current value resides.

## Aggregate Functions

DLC aggregate functions are identical to Hive aggregate functions.

For details about Hive collection functions, refer to the [CollectionFunctions](#) page on the Apache Hive website.

For DLC aggregate functions, see the [Aggregate Functions](#) document on the Tencent Cloud website.

| Hive 3.1 Function Name | Function Description                                 | DLC Function Name | Difference Description | Usage Reference                              |
|------------------------|------------------------------------------------------|-------------------|------------------------|----------------------------------------------|
| size                   | Returns the size of a map/array.                     | size              | No difference          | select size(str_to_map('a:1,b:2'));          |
| map_keys               | Returns the list of keys in a map.                   | map_keys          | No difference          | select map_keys(str_to_map('a:1,b:2'));      |
| map_values             | Returns the list of values in a map.                 | map_values        | No difference          | select map_values(str_to_map('a:1,b:2'));    |
| array_contains         | Whether an array contains a value                    | array_contains    | No difference          | select array_contains(split('a,b',','),'a'); |
| sort_array             | Sorting the elements in an array in ascending order. | sort_array        | No difference          | select sort_array(split('1,3,2',','));       |

## Type Conversion Functions

| Hive 3.1 Function Name | Function Description | DLC Function Name | Difference Description | Usage Reference |
|------------------------|----------------------|-------------------|------------------------|-----------------|
|------------------------|----------------------|-------------------|------------------------|-----------------|



|        |                                                                                                                                                                                                                                                                                                         | on<br>Name |                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                             |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| binary | Converts the input parameter to a binary array.                                                                                                                                                                                                                                                         | binary     | <ul style="list-style-type: none"> <li>Hive 3.1 input parameters only support string and binary types. DLC supports string, int, long, and binary input parameters.</li> <li>In Hive 3.2, output parameters are converted to strings for display. In DLC, output parameters are binary arrays and are not converted to strings.</li> </ul> | <pre>select binary('testString') select binary(1) (HIVE does not support this). select binary(inputCol) from inputTable (HIVE does not support this).</pre> |
| cast   | <ul style="list-style-type: none"> <li>Cast the input parameter to the specified type.</li> <li>An error occurs if the given expression or value cannot be cast to the specified type. For example, the error for converting a string to long is: Cast function cannot convert value of type</li> </ul> | cast       | Hive3.1 returns NULL for failed casts, while DLC reports an error.                                                                                                                                                                                                                                                                         | <pre>select cast('10' as int)select cast(inputCol as int) from inputTable</pre>                                                                             |

VARCHAR(65536)  
 ) to type LONG.

Time Functions

| Hive 3.1<br>Function<br>Name | Function<br>Description                                                                                                                                                                                                                                                           | DLC<br>Cor<br>res<br>pon<br>ding<br>Fun<br>ctio<br>n<br>Na<br>me | Difference<br>Description | Usage Reference                                                                      |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|---------------------------|--------------------------------------------------------------------------------------|
| from_unix<br>time            | Converts a numeric Unix timestamp (representing the number of seconds counted from 1970-01-01 00:00:00 UTC) into a string function of a specified format. The default output timestamp format is yyyy-MM-dd HH:mm:ss, and the time zone is the one defined by the current system. | from_unix_time                                                   | No difference             | select<br>from_unixtime(12458456)<br>select from_unixtime(100, 'yyyyMMdd HH:mm:ssZ') |
| unix_time<br>stamp           | Specifies a timestamp, or defaults to the current time, and returns the                                                                                                                                                                                                           | unix_timestamp                                                   | No difference             | select unix_timestamp()<br>select<br>unix_timestamp('2023-04-12 00:00:00')           |

|         |                                                                                                                                           |         |                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                            |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | number of seconds from 1970-01-01 00:00:00 UTC to the specified time.                                                                     |         |                                                                                                                                                                                                                                                                                                                       | <pre>select unix_timestamp('2023-04-12', 'yyyy-MM-dd')</pre>                                                                                                                                               |
| to_date | Specifies a timestamp and obtains the date to which it belongs. For example, the date to which 2023-04-12 13:14:20 belongs is 2023-04-12. | to_date | <ul style="list-style-type: none"><li>The HIVE to_date function only accepts input in the specified format yyyy-MM-dd HH:mm:ss. Inputs in any other format uniformly return NULL.</li><li>DLC can accept an additional parameter to specify the data format, allowing for more flexible input time formats.</li></ul> | <pre>select to_date('2023-04-12 19:41:23') select to_date('20230412 19:41', 'yyyy-MM-dd HH:mm') (Not supported in HIVE). select to_date('20230412 19:41', 'yyyyMMdd HH:mm') (Not supported in HIVE).</pre> |
| year    | Specifies a timestamp and obtains the year to which it belongs. For example, the day to                                                   | year    | <ul style="list-style-type: none"><li>HIVE input does not support the yyyy or yyyy-MM formats,</li></ul>                                                                                                                                                                                                              | <pre>select year('2023-04-12 19:41:25') select year('2023-04') (Not supported in HIVE).</pre>                                                                                                              |

|         |                                                                                                                                               |         |                                                                                                                                                                                                                      |                                                                                                                    |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
|         | which 2023-04-12 13:14:20 belongs is 2023.                                                                                                    |         | <p>but supports all other time or date formats.</p> <ul style="list-style-type: none"> <li>DLC, however, supports the yyyy or yyyy-MM formats.</li> </ul>                                                            |                                                                                                                    |
| quarter | <p>Specifies a timestamp and obtains the quarter to which it belongs. For example, the quarter to which 2023-04-12 13:14:20 belongs is 2.</p> | quarter | <ul style="list-style-type: none"> <li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li> <li>DLC, however, supports the yyyy or yyyy-MM formats.</li> </ul> | <pre>select quarter('2023-04-12 19:41')</pre> <pre>select quarter('2023-04')</pre> <p>(Not supported in HIVE).</p> |
| month   | <p>Specifies a timestamp and obtains the month to which it belongs. For example, the month to which 2023-04-12 13:14:20 belongs is 4.</p>     | month   | <ul style="list-style-type: none"> <li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other</li> </ul>                                                                                    | <pre>select month('2023-04-12 19:41')</pre> <pre>select month('2023 -04')</pre> <p>(Not supported in HIVE).</p>    |

|                 |                                                                                                                                                                        |                   |                                                                                                                                                                                                                                                                                                |                                                                                                                                       |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
|                 |                                                                                                                                                                        |                   | <p>time or date formats.</p> <ul style="list-style-type: none"><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• When the month in a time format exceeds 12, HIVE calculates the difference from the last month and takes the remainder, whereas DLC returns NULL.</li></ul> |                                                                                                                                       |
| day/dayof month | <p>Specifies a timestamp and obtains the day to which it belongs.</p> <p>For example, the month to which 2023-04-12 13:14:20 belongs is the 12th day of the month.</p> | day /day of month | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>• DLC, however, supports</li></ul>                                                                                                       | <p>select day('2023') (Not supported in HIVE).</p> <p>select dayofmonth('2023-04-12 19:41')</p> <p>select day('2023-04-12 19:41')</p> |

|      |                                                                                                                                          |      |                                                                                                                                                                                                                                                     |                                                                                               |
|------|------------------------------------------------------------------------------------------------------------------------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
|      |                                                                                                                                          |      | <p>the yyyy or yyyy-MM formats.</p> <ul style="list-style-type: none"><li>When the day in a time format exceeds the number of days in that month, HIVE calculates the remainder against the month's total days, whereas DLC returns NULL.</li></ul> |                                                                                               |
| hour | <p>Specifies a timestamp and obtains the hour to which it belongs. For example, the hour to which 2023-04-12 13:14:20 belongs is 13.</p> | hour | <ul style="list-style-type: none"><li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>DLC, however, supports the yyyy or yyyy-</li></ul>                                              | <p>select hour('2023-04-12 19:41')</p> <p>select hour('2023-04') (Not supported in HIVE).</p> |

|        |                                                                                                                                              |        |                                                                                                                                                                                                                                                                                              |                                                                                                          |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
|        |                                                                                                                                              |        | <p>MM formats.</p> <ul style="list-style-type: none"><li>• When the hour in a time format exceeds 23, HIVE calculates the remainder against 23, whereas DLC returns NULL.</li></ul>                                                                                                          |                                                                                                          |
| minute | <p>Specifies a timestamp and obtains the minute to which it belongs. For example, the minute to which 2023-04-12 13:14:20 belongs is 14.</p> | minute | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• When the minute in a time format exceeds 59, HIVE calculates</li></ul> | <p>select minute('2023-04-12 00:41')</p> <p>select minute('2023-04')</p> <p>(Not supported in HIVE).</p> |

|        |                                                                                                                                                          |        |                                                                                                                                                                                                                                                                                                                                                  |                                                                                                             |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
|        |                                                                                                                                                          |        | the remainder against 60, whereas DLC returns NULL.                                                                                                                                                                                                                                                                                              |                                                                                                             |
| second | <p>Specifies a timestamp and obtains the second to which it belongs. For example, the month to which 2023-04-12 13:14:20 belongs is the 20th second.</p> | second | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• When the second in a time format exceeds 59, HIVE calculates the remainder against 60, whereas DLC returns NULL.</li></ul> | <p>select second('2023-04-12 00:41:24')</p> <p>select second('2023-04')</p> <p>(Not supported in HIVE).</p> |



|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                         |                                                                                                                                                                                                                       |                                                                                                           |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>weekofyear</code> | <p>Specifies a timestamp and returns the week number of the year to which it belongs.</p> <p>For example, 2023-04-12 13:14:20 is the 15th week of 2023.</p> <p>Note that the first Monday of the year is considered the start of the first week. If a datetime is before the first Monday of the year, it is considered the 52nd week of the previous year. For example, 2023-01-01 is a Sunday, so <code>weekofyear('2023-01-01')</code> returns 52, which is the 52nd week of the previous year, while <code>weekofyear('2023-01-02')</code> returns 1, which is the first week of the current year.</p> | <code>weekofyear</code> | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li></ul> | <pre>select weekofyear('2023-04-12 00:41:25') select weekofyear('2023-04') (Not supported in HIVE).</pre> |
| <code>extract</code>    | <p>Extracts the specified field</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <code>extract</code>    | <ul style="list-style-type: none"><li>• Adjusting precision</li></ul>                                                                                                                                                 | <pre>select extract(DAY FROM DATE '2023-04-12')</pre>                                                     |

from the input source datetime or interval.

is currently not supported in the HIVE and DLC versions.

- When a DATE or TIMESTAM P identifier is explicitly specified, DLC reports an error if the date or time format does not match, whereas HIVE returns 0.
- When calculating

INTERVAL intervals, if a field is not in the source, HIVE automatically calculates the remainder, whereas DLC reports an error. For

```
select extract(second from
TIMESTAMP '2023-04-12
00:41:25')
select extract(month from
interval '23-1' YEAR TO
MONTH)
select extract(month from
interval '23' MONTH)
select extract(year from
interval '23' MONTH) (Not
supported in DLC).
select extract(hour from
interval '23 13:23:34.34784'
DAY TO SECOND)
```

example, when extracting the year field from an interval of 24 months, HIVE returns 2 years, while DLC reports that the field does not exist.

datediff

Calculates the number of days between startDate and endDate. If startDate is later than endDate, a negative number is returned. Returns NULL if any input is NULL.

datediff

- HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.
- DLC, however, supports the yyyy or yyyy-MM formats.

select datediff('2021', '2022-10') (Not supported in HIVE).  
select datediff('2023-04-13', '2022-04-13 11:00:00')

from\_utc\_timestamp

Converts a given UTC timestamp to the time in a specified timeZone.

from\_utc\_timestamp

- HIVE input does not support the yyyy or yyyy-MM formats,

select from\_utc\_timestamp('2023-04', 'Asia/Seoul')  
select from\_utc\_timestamp('2023-04-12 15:00:00', 'Asia/Shanghai')

|  |                                    |  |                                                                                                                                                                                                                                                                                                                                                                                              |  |
|--|------------------------------------|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
|  | Returns NULL if any input is NULL. |  | <p>but supports all other time or date formats.</p> <ul style="list-style-type: none"><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• Hive returns a time format, represented as a string in the yyyy-MM-dd HH:mm:ss format.</li><li>• DLC, however, returns a UTC-standard representation, for example, yyyy-MM-ddTHH:mm:ss+08:00 for the Shanghai time zone.</li></ul> |  |
|--|------------------------------------|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

|                  |                                                                                                      |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                         |
|------------------|------------------------------------------------------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| to_utc_timestamp | <p>Converts a given timestamp in a specified timeZone to UTC. Returns NULL if any input is NULL.</p> | to_utc_timestamp | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• Hive returns a time format, represented as a string in the yyyy-MM-dd HH:mm:ss format.</li><li>• DLC, however, returns a UTC-standard representation, for example, yyyy-MM-ddTHH:mm:ss+08:00 for the</li></ul> | <pre>select to_utc_timestamp('2023-04-12 15:00:00', 'Asia/Shanghai') select to_utc_timestamp('2023-04', 'Asia/Shanghai') (Not supported in HIVE).</pre> |
|------------------|------------------------------------------------------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|

Shanghai  
time zone.

date\_add

Performs date addition. Given a startDate, returns the date after adding numDays. numDays can be a negative number. Returns NULL if any input is NULL.

date\_add

- HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.
- DLC can support the yyyy or yyyy-MM formats.

select date\_add('2023-04-12 15:00:00', 1)  
select date\_add('2023-04', -1) (Not supported in HIVE).

date\_sub

Performs date subtraction. Given a startDate, it returns the date after numDays are subtracted. numDays can be a negative number. Returns NULL if any input is NULL.

date\_sub

- HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.
- DLC, however, supports the yyyy or yyyy-MM formats.

select date\_sub('2023-04-12 15:00:00', 1)  
select date\_sub('2023-04', -1) (Not supported in HIVE).

|                   |                                                                                                                                     |                   |                                                                                                                                                                                                                      |                                                                                                                  |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| current_date      | Obtains the current date.                                                                                                           | current_date      | No difference                                                                                                                                                                                                        | select current_date()                                                                                            |
| current_timestamp | Obtains the current time.                                                                                                           | current_timestamp | No difference                                                                                                                                                                                                        | select current_timestamp()                                                                                       |
| add_months        | Calculates the date after a specified number of months are added to a given date (or timestamp). Returns NULL if the input is NULL. | add_months        | <ul style="list-style-type: none"> <li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li> <li>DLC, however, supports the yyyy or yyyy-MM formats.</li> </ul> | <pre>select add_months('2023-04-12 15:00:00', 1) select add_months('2023-04', -1) (Not supported in HIVE).</pre> |
| last_day          | Calculates the last day of the month for a specified date. Returns NULL if the input is NULL.                                       | last_day          | <ul style="list-style-type: none"> <li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or</li> </ul>                                                                            | <pre>select last_day('2023-04-12 15:00:00') select last_day('2023-04') (Not supported in HIVE).</pre>            |

|          |                                                                                                                                                                                                              |          |                                                                                                                                                                                                                                  |                                                                                                                                    |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
|          |                                                                                                                                                                                                              |          | <p>date formats.</p> <ul style="list-style-type: none"> <li>DLC, however, supports the yyyy or yyyy-MM formats.</li> </ul>                                                                                                       |                                                                                                                                    |
| next_day | <p>Calculates the date of the first specified day_of_week after a given date. The day_of_week indicates the day of the week, with valid values from Monday to Sunday. Returns NULL if any input is NULL.</p> | next_day | <ul style="list-style-type: none"> <li>The yyyy or yyyy-MM formats are not supported for HIVE input, while all other time or date formats are supported.</li> <li>DLC, however, supports the yyyy or yyyy-MM formats.</li> </ul> | <pre>select next_day('2023-04-12 15:00:00')</pre> <pre>select next_day('2023-04')</pre> <p>(Not supported in HIVE).</p>            |
| trunc    | <p>Calculates the date of the first day after a specified date is truncated by a format (such as year, quarter, month, and week), such as the first day of the quarter, the first day of the</p>             | trunc    | <ul style="list-style-type: none"> <li>HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or date formats.</li> </ul>                                                                          | <pre>select trunc('2023-04-12 15:00:00', 'MONTH')</pre> <pre>select trunc('2023-04', 'YEAR')</pre> <p>(Not supported in HIVE).</p> |



|                |                                                                                                                                                                                                                                                                           |                |                                                                                                                                                          |                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | month, or the first day of the week.                                                                                                                                                                                                                                      |                | <ul style="list-style-type: none"><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li><li>• HIVE does not support the WEEK function.</li></ul> |                                                                                                                                                                 |
| -              | Calculates the time of the first day after a specified date is truncated by a format (such as year, quarter, month, week, and so on), for example, 00:00:00 on the first day of the quarter. The difference from the trunc method is that date_trunc returns a timestamp. | date_trunc     | HIVE does not support this function. Refer to the trunc function.                                                                                        | <pre>select date_trunc('MM', '2023-04-12 15:00:00') select date_trunc('SECOND', '2023-04-12 15:00:60')</pre>                                                    |
| months_between | Calculates the number of months between date1 and date2. If date1 < date2, a negative number is returned. Note that during                                                                                                                                                | months_between | <ul style="list-style-type: none"><li>• HIVE input does not support the yyyy or yyyy-MM formats, but supports all other time or</li></ul>                | <pre>select months_between('2023-04-12 15:00:00', '2023-04-12 15:00:00') select months_between('2023-04', '2023-04-12 15:00:00') (Not supported in HIVE).</pre> |

|                    |                                                                                                                                                                                                                                                                               |                    |                                                                                                                                                |                                                                                                                |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
|                    | <p>calculation, each month is estimated as 31 days, meaning the denominator is 31. The numerator is the actual time difference between the two timestamps, accurate to the millisecond level. The final result is a decimal. Returns NULL if any input parameter is NULL.</p> |                    | <p>date formats.</p> <ul style="list-style-type: none"><li>• DLC, however, supports the yyyy or yyyy-MM formats.</li></ul>                     |                                                                                                                |
| <p>date_format</p> | <p>Formats an input date according to a specified formatting template fmt (refer to <a href="#">Datetime Patterns for Formatting and Parsing</a>). Returns NULL if any input parameter is NULL.</p>                                                                           | <p>date_format</p> | <p>When the date time or date format is incorrect, or the fmt template format is incorrect, HIVE returns NULL, while DLC reports an error.</p> | <pre>select date_format('2023-04-12 15:00:00', 'y') select date_format('2023-04-12', 'yyyy-MM-dd HH:mm')</pre> |

Conditional Functions

| Hive 3.1 Function Name | Function Description | DLC Function Name | Difference Description | Usage Reference |
|------------------------|----------------------|-------------------|------------------------|-----------------|
|------------------------|----------------------|-------------------|------------------------|-----------------|

|           |                                                                                   |           |                                                                                                                                                                                                |                                                                                                                                                  |
|-----------|-----------------------------------------------------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| if        | Returns valueTrue if the condition is true; otherwise, returns valueFalseOr Null. | if        | When the condition is NULL, HIVE treats it as false, while DLC reports an error.                                                                                                               | select if(1<2, 12, 'false')                                                                                                                      |
| isnull    | Returns true if a is NULL; otherwise, returns false.                              | isnull    | No difference                                                                                                                                                                                  | select isnull('false')                                                                                                                           |
| isnotnull | Returns true if a is not NULL; otherwise, returns false.                          | isnotnull | No difference                                                                                                                                                                                  | select isnotnull('false')                                                                                                                        |
| nvl       | Returns the second parameter if the value is NULL.                                | nvl       | No difference                                                                                                                                                                                  | select nvl(NULL, 'false')<br>select nvl(1, 'false')                                                                                              |
| coalesce  | Returns the first non-NULL parameter.                                             | coalesce  | No difference                                                                                                                                                                                  | select coalesce(NULL, 'false')                                                                                                                   |
| case/when | Matching and judging                                                              | case/when | When the types of the matched selections differ, HIVE only supports implicit conversion for integer and string types, while DLC also supports implicit conversion for boolean and other types. | select CASE 'c'<br>WHEN 'a' THEN 1<br>WHEN 'b' THEN 2<br>ELSE 0 END;<br>select CASE WHEN<br>1 > 0 THEN 1 WHEN<br>2 > 0 THEN 2.0<br>ELSE 1.2 END; |
| nullif    | Returns null if parameter a equals b;                                             | nullif    | HIVE does not support the first parameter being                                                                                                                                                | select nullif(1, 'test');                                                                                                                        |

|                 |                                                                                                |                 |                                                                |                                                                  |
|-----------------|------------------------------------------------------------------------------------------------|-----------------|----------------------------------------------------------------|------------------------------------------------------------------|
|                 | otherwise,<br>returns a.                                                                       |                 | NULL, while DLC can<br>support any<br>parameter being<br>NULL. | select nullif(NULL,<br>'test'); (HIVE does<br>not support this). |
| assert_tr<br>ue | Throws an<br>exception if<br>the<br>"condition" is<br>not true;<br>otherwise,<br>returns null. | assert<br>_true | No difference                                                  | select<br>assert_true(1>0)                                       |

String Functions

| Hive 3.1 Function Name | Function Description                                                                                                                                                                                                                                                                                                                              | DLC Corresponding Function Name | Difference Description                                                            | Usage Reference                                 |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------|
| ascii                  | Returns the numeric value of the first character in str.                                                                                                                                                                                                                                                                                          | ascii                           | No difference                                                                     | select ascii('222');                            |
| base64                 | Converting a parameter from binary to a base64 string.                                                                                                                                                                                                                                                                                            | base64                          | No difference                                                                     | select base64('tencent');                       |
| character_length       | Returns the number of UTF-8 characters contained in str.                                                                                                                                                                                                                                                                                          | character_length                | No difference                                                                     | select char_length(binary('tencent'));          |
| chr                    | Returns the binary character equivalent to A.                                                                                                                                                                                                                                                                                                     | chr                             | No difference                                                                     | select chr(65);                                 |
| concat                 | Returns the string or byte sequence produced by concatenating the passed strings or bytes in order.                                                                                                                                                                                                                                               | concat                          | No difference                                                                     | select concat('Spark', 'SQL');                  |
| context_ngrams         | Given a contextual N-grams, returns the top k contextual N-grams from a set of tokenized sentences.                                                                                                                                                                                                                                               | context_ngrams                  | No difference                                                                     | -                                               |
| concat_ws              | Returns a string separated by sep.                                                                                                                                                                                                                                                                                                                | concat_ws                       | No difference                                                                     | select concat_ws(' ', 'tencent', 'dlc');        |
| decode                 | Decodes the first argument into a string using the provided character set (one of "US-ASCII", "ISO-8859-1", "UTF-8", "UTF-16BE", "UTF-16LE", "UTF-16"). If any argument is null, the result will also be null.                                                                                                                                    | decode                          | No difference                                                                     | select decode(encode('abc', 'utf-8'), 'utf-8'); |
| elt                    | Returns the string corresponding to the index number. For example, elt(2,'hello','world') returns 'world'. If N is less than 1 or greater than the number of arguments, returns NULL.                                                                                                                                                             | elt                             | No difference                                                                     | select elt(1, 'scala', 'java');                 |
| encode                 | Encodes the first argument into binary using the provided character set (one of "US-ASCII", "ISO-8859-1", "UTF-8", "UTF-16BE", "UTF-16LE", "UTF-16"). If any argument is Null, the result will also be Null.                                                                                                                                      | encode                          | No difference                                                                     | select encode('abc', 'utf-8');                  |
| field                  | field(val T, val1 T, val2 T, val3 T,...)<br>Returns the index position of val within the argument list val1, val2, val3,..., returning 0 if not found. For example, field('world', 'say', 'hello', 'world') returns 3.<br>This function supports all basic data types, and parameters are compared using str.equals(x). Returns 0 if val is NULL. | field                           | Supported only by the presto engine in DLC.                                       | select field('world', 'say', 'hello', 'world'); |
| find_in_set            | Returns the index (starting from 1) of the given string str within the comma-separated list atr_array. Returns 0 if the string is not found or if str contains a comma.                                                                                                                                                                           | find_in_set                     | No difference                                                                     | select find_in_set('ab', 'abc,b,ab,c,def');     |
| format_number          | Formats the input argument into '#,###,###.##', rounds it to D decimal places, and returns the result as a string.                                                                                                                                                                                                                                | format_number                   | No difference                                                                     | select format_number(12332.123456, 4);          |
| get_json_object        | Extracts a json object.                                                                                                                                                                                                                                                                                                                           | get_json_object                 | No difference                                                                     | select get_json_object('{ "a": "b" }', '\$.a'); |
| in_file                | in_file(string str, string filename)<br>Returns true if the string str appears as an entire line in the file filename.                                                                                                                                                                                                                            | Not supported.                  | -                                                                                 | -                                               |
| instr                  | Returns the index (starting from 1) of the first occurrence of substr in str.                                                                                                                                                                                                                                                                     | instr                           | No difference                                                                     | select instr('SparkSQL', 'SQL');                |
| length                 | Returns the length of a string.                                                                                                                                                                                                                                                                                                                   | length                          | No difference                                                                     | select length('Spark SQL');                     |
| locate                 | Returns the position of the first occurrence of substr in str after position pos.                                                                                                                                                                                                                                                                 | locate                          | No difference                                                                     | select locate('bar', 'foobarbar');              |
| lower                  | Returns the string produced by converting all characters in B to lowercase.                                                                                                                                                                                                                                                                       | lower                           | No difference                                                                     | select lower('TENCENT');                        |
| lpad                   | lpad(string str, int len, string pad)<br>Left-pads the string str with the character pad to a length of len. If the length of str is greater than len, the return value is truncated to len characters. If the padding character pad is empty, the return value is null.                                                                          | lpad                            | In the DLC sparksql engine, pad is an optional parameter and defaults to a space. | select lpad('hi', 5, '??');                     |
|                        | Returns the string produced by trimmina                                                                                                                                                                                                                                                                                                           |                                 |                                                                                   |                                                 |

|                |                                                                                                                                                                                                                                                                            |                |                                                                                                                                                                                                                       |                                                                   |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| ltrim          | whitespace from the beginning (left side) of the input string.                                                                                                                                                                                                             | ltrim          | No difference                                                                                                                                                                                                         | select ltrim(' SparkSQL ');                                       |
| ngrams         | Returns the top k N-grams from a set of tokenized sentences.                                                                                                                                                                                                               | ngrams         | No difference                                                                                                                                                                                                         | –                                                                 |
| octet_length   | Returns the byte length of string data or the number of bytes in binary data.                                                                                                                                                                                              | octet_length   | No difference                                                                                                                                                                                                         | select octet_length('Spark SQL');                                 |
| parse_url      | Extracts the path from a url.                                                                                                                                                                                                                                              | parse_url      | No difference                                                                                                                                                                                                         | select parse_url('http://spark.apache.org/path?query=1', 'HOST'); |
| printf         | Returns the formatted string from a printf-style format string.                                                                                                                                                                                                            | printf         | No difference                                                                                                                                                                                                         | select printf("Hello World %d %s", 100, "days");                  |
| quote          | Returns a quoted string.                                                                                                                                                                                                                                                   | Not supported. | –                                                                                                                                                                                                                     | –                                                                 |
| regexp_extract | Extracts the first string in str that matches the regexp expression and corresponds to the regex group index idx.                                                                                                                                                          | regexp_extract | No difference                                                                                                                                                                                                         | select regexp_extract('100-200', '(\\d+)-(\\d+)', 1);             |
| regexp_replace | regexp_replace(string INITIAL_STRING, string PATTERN, string REPLACEMENT)<br>Returns the string obtained by replacing all substrings in INITIAL_STRING that match the Java regular expression syntax defined in PATTERN with instances of REPLACEMENT.                     | regexp_replace | In the DLC sparksql engine, a fourth optional parameter, position, can be added. Position is a positive integer literal that indicates the starting position for searching within the string. The default value is 1. | select regexp_replace('100-200', '(\\d+)', 'num');                |
| repeat         | Returns the string produced by repeating a given string n times.                                                                                                                                                                                                           | repeat         | No difference                                                                                                                                                                                                         | select repeat('123', 2);                                          |
| replace        | replace(string A, string OLD, string NEW)<br>Returns the string obtained by replacing all non-overlapping occurrences of OLD in string A with NEW.                                                                                                                         | replace        | In the sparksql engine, the third parameter, NEW, is optional and defaults to an empty string.                                                                                                                        | select replace('ABCabc', 'abc', 'DEF');                           |
| reverse        | reverse(string A)<br>Returns the reversed string.                                                                                                                                                                                                                          | reverse        | In the sparksql engine, the parameter can be an array.                                                                                                                                                                | select reverse('Spark SQL');                                      |
| rpadd          | rpadd(string str, int len, string pad)<br>Right-pads the string str with the character pad to a length of len. If the length of str is greater than len, the return value is truncated to len characters. If the padding character pad is empty, the return value is null. | rpadd          | In the sparksql engine, pad is an optional parameter and defaults to a space.                                                                                                                                         | select rpadd('hi', 5, '??');                                      |
| rtrim          | Returns the string produced by trimming whitespace from the end (right side) of the input string.                                                                                                                                                                          | rtrim          | No difference                                                                                                                                                                                                         | select rtrim(' SparkSQL ');                                       |
| sentences      | Splits the input string str into an array of words.                                                                                                                                                                                                                        | sentences      | No difference                                                                                                                                                                                                         | select sentences('Hi there! Good morning.');                      |
| space          | Returns a string consisting of n spaces.                                                                                                                                                                                                                                   | space          | No difference                                                                                                                                                                                                         | select concat(space(2), '1');                                     |
| split          | split(string str, string pat)<br>Splits the string str around the regular expression pat (pat is a regular expression).                                                                                                                                                    | split          | In the DLC sparksql engine, there is a third optional parameter, limit. limit can be used to restrict the length of the returned array.                                                                               | select split('oneAtwoBthreeC', '[ABC]');                          |
| str_to_map     | Creates a map by splitting the input parameters into key/value pairs using delimiters.                                                                                                                                                                                     | str_to_map     | No difference                                                                                                                                                                                                         | select str_to_map('a:1,b:2,c:3', ',', ':');                       |
| substr         | Returns a substring of str starting at pos with a length of len, or a slice of a byte array starting at pos with a length of len.                                                                                                                                          | substr         | No difference                                                                                                                                                                                                         | select substr('Spark SQL', 5);                                    |
| substring      | Returns a substring of str starting at pos with a length of len, or a slice of a byte array starting at pos with a length of len.                                                                                                                                          | substring      | No difference                                                                                                                                                                                                         | select substring('Spark SQL', 5);                                 |

|                                 |                                                                                                                                                                                                                                                                                                                                         |                                 |                                                                     |                                                                 |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|---------------------------------------------------------------------|-----------------------------------------------------------------|
| substring_index                 | Returns a substring from str before the count-th occurrence of delim. If count is positive, returns everything to the left of the final delimiter (counting from the left). If count is negative, returns everything to the right of the final delimiter (counting from the right). The function is case-sensitive when matching delim. | <a href="#">substring_index</a> | No difference                                                       | <pre>select substring_index('cloud.tencent.com', '.', 2);</pre> |
| translate                       | Converts the input string by replacing characters in the from string with the corresponding characters in the to string.                                                                                                                                                                                                                | <a href="#">translate</a>       | No difference                                                       | <pre>select translate('AaBbCc', 'abc', '123');</pre>            |
| trim                            | Returns the string produced by trimming whitespace from both ends of the input string.                                                                                                                                                                                                                                                  | <a href="#">trim</a>            | The DLC SparkSQL engine supports more complex trimming expressions. | <pre>select trim(' SparkSQL ');</pre>                           |
| unbase64                        | Converting the str from a base64 string to binary.                                                                                                                                                                                                                                                                                      | <a href="#">unbase64</a>        | No difference                                                       | <pre>select unbase64('U3BhcmsgU1FM');</pre>                     |
| upper                           | Returns the str with all characters converted to uppercase.                                                                                                                                                                                                                                                                             | <a href="#">upper</a>           | No difference                                                       | <pre>select upper('tencent');</pre>                             |
| ucase                           | Returns the str with all characters converted to uppercase.                                                                                                                                                                                                                                                                             | <a href="#">ucase</a>           | No difference                                                       | <pre>select ucase('SparkSQL');</pre>                            |
| initcap                         | Converts the first letter of each word to uppercase and all other letters to lowercase.                                                                                                                                                                                                                                                 | <a href="#">initcap</a>         | No difference                                                       | <pre>select initcap('sPark sql');</pre>                         |
| levenshtein(string A, string B) | Returns the Levenshtein distance between two given strings.                                                                                                                                                                                                                                                                             | <a href="#">levenshtein</a>     | No difference                                                       | <pre>select levenshtein('kitten', 'sitting');</pre>             |
| soundex(string A)               | Returns the Soundex encoding of a string.                                                                                                                                                                                                                                                                                               | <a href="#">soundex</a>         | No difference                                                       | <pre>select soundex('Miller');</pre>                            |

## Data Masking Functions

| Hive 3.1 Function Name | Function Description                                                                                                                                     | DLC Corresponding Function Name | Difference Description | Usage Reference |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|------------------------|-----------------|
| mask                   | <code>mask(string str[, string upper[, string lower[, string number]]])</code><br>Returns a masked version of str.                                       | Not supported.                  | —                      | —               |
| mask_first_n           | <code>mask_first_n(string str[, int n])</code><br>Returns a masked version of str, where the first n values are replaced by masks.                       | Not supported.                  | —                      | —               |
| mask_last_n            | <code>mask_last_n(string str[, int n])</code><br>Returns a masked version of str, where the last n values are replaced by masks.                         | Not supported.                  | —                      | —               |
| mask_show_first_n      | <code>mask_show_first_n(string str[, int n])</code><br>Returns a masked version of str, where the first n characters are displayed without being masked. | Not supported.                  | —                      | —               |

|                  |                                                                                                                                            |                |   |   |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------|----------------|---|---|
| mask_show_last_n | mask_show_last_n(string str[, int n])<br>Returns a masked version of str, where the first n characters are displayed without being masked. | Not supported. | – | – |
| mask_hash        | mask_hash(string char varchar str)<br>Returns a hash value based on str.                                                                   | Not supported. | – | – |

## Other Functions

| Hive 3.1 Function Name | Function Description                                                                                                                                                                           | DLC Corresponding Function Name | Difference Description                                                                 | Usage Reference                                                                    |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| java_method            | a synonym for reflect                                                                                                                                                                          | Not supported.                  | –                                                                                      | –                                                                                  |
| reflect                | Invoking Java methods by matching parameter signatures using reflection.                                                                                                                       | Not supported.                  | –                                                                                      | –                                                                                  |
| hash                   | Returning the hash value of the parameter(s)                                                                                                                                                   | hash                            | Different engines may yield different results due to inconsistent calculation methods. | select hash('tencent', array(123), 2);                                             |
| current_user           | Returns the current user.                                                                                                                                                                      | current_user                    | No difference                                                                          | select current_user();                                                             |
| logged_in_user         | Returns the current username from the session state. This is the username provided when a connection to Hive is made.                                                                          | Not supported.                  | –                                                                                      | –                                                                                  |
| current_database       | Returns the name of the current database.                                                                                                                                                      | Not supported.                  | –                                                                                      | –                                                                                  |
| md5                    | Returns the MD5 128-bit checksum as a hexadecimal string.                                                                                                                                      | md5                             | No difference                                                                          | select md5('tencent');                                                             |
| sha1                   | Returns the SHA-1 hash value of the input parameter(s) as a hexadecimal string.                                                                                                                | sha1                            | No difference                                                                          | select sha1('tencent');                                                            |
| sha                    | Returns the SHA-1 hash value of the input parameter(s) as a hexadecimal string.                                                                                                                | sha                             | No difference                                                                          | select sha('tencent');                                                             |
| crc32                  | Computing the cyclic redundancy check value of the expression using the CRC32 algorithm                                                                                                        | crc32                           | No difference                                                                          | select crc32('tencent');                                                           |
| sha2                   | Returns the checksum of the SHA-2 family for expr as a hexadecimal string. Supports SHA-224, SHA-256, SHA-384, and SHA-512. A bit length of 0 equals 256.                                      | sha2                            | No difference                                                                          | select sha2('tencent', 256);                                                       |
| aes_encrypt            | aes_encrypt(input string/binary, key string/binary)<br>Encrypting using aes                                                                                                                    | aes_encrypt                     | This function is not supported by the sparksql engine in DLC.                          | select hex(aes_encrypt('tencent', '0000111122223333'));                            |
| aes_decrypt            | aes_decrypt(input binary, key string/binary)<br>Decrypting using AES                                                                                                                           | aes_decrypt                     | This function is not supported by the sparksql engine in DLC.                          | select aes_decrypt(unhex('B99B99CE3359A736DBB9811ED8815C01'), '0000111122223333'); |
| version                | version()<br>Returns the engine version.                                                                                                                                                       | version                         | sparksql returns the spark version, while presto returns the hive version.             | select version();                                                                  |
| surrogate_key          | surrogate_key([write_id_bits, task_id_bits])<br>Automatically generates a numeric ID when data is inserted into a table. Can only be used as the default value for ACID or insert-only tables. | Not supported.                  | –                                                                                      | –                                                                                  |





# Overview of Standard Spark Syntax

Last updated: 2026-05-21 14:37:05

The kernel of the standard Spark engine is developed in-house based on the Spark 3.2 version and is compatible with native Spark syntax and behavior. For detailed syntax, see the Spark syntax documentation.

## DDL Syntax

### Database-Related Syntax

| Purpose                                           | Syntax                          |
|---------------------------------------------------|---------------------------------|
| Creating a database                               | CREATE DATABASE                 |
| Displaying all databases defined in the metadata. | SHOW DATABASES                  |
| Viewing database properties                       | DESCRIBE DATABASE               |
| Modifying database properties                     | ALTER DATABASE SET DBPROPERTIES |
| Delete a database.                                | DROP DATABASE                   |

### Data Table-Related Syntax

| Purpose                                            | Syntax                             |
|----------------------------------------------------|------------------------------------|
| Creating a data table                              | CREATE TABLE                       |
| Querying data table creation information           | SHOW CREATE TABLE                  |
| Querying table properties                          | SHOW TBLPROPERTIES                 |
| Querying all tables in a database                  | SHOW TABLES                        |
| Viewing data table column information and metadata | DESCRIBE TABLE                     |
| Adding columns to a data table                     | ALTER TABLE ADD COLUMNS            |
| Adding a new column to a data table                | ALTER TABLE ADD COLUMN AFTER/FIRST |

|                                                                        |                                                |
|------------------------------------------------------------------------|------------------------------------------------|
| Renaming a field                                                       | ALTER TABLE ... RENAME COLUMN                  |
| Dropping a column from a data table                                    | ALTER TABLE DROP COLUMN                        |
| Adding partition information to a data table                           | ALTER TABLE ADD PARTITION                      |
| Listing table partitions                                               | SHOW PARTITIONS                                |
| Dropping partition information from a data table                       | ALTER TABLE DROP PARTITION                     |
| Adding a partition field to an Iceberg table                           | ALTER TABLE ADD PARTITION FIELD                |
| Dropping a partition field from an Iceberg table                       | ALTER TABLE DROP PARTITION FIELD               |
| Modifying data table properties                                        | ALTER TABLE SET TBLPROPERTIES                  |
| Changing data table storage location                                   | ALTER TABLE SET LOCATION                       |
| Modifying the sort order of table data                                 | ALTER TABLE ... WRITE ORDERED BY               |
| Modifying the distribution policy for a partitioned table              | ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION |
| Updating partition information (not supported for Iceberg table V2)    | MSCK REPAIR TABLE                              |
| Dropping a metadata table                                              | DROP TABLE                                     |
| Displaying the logical or physical plan for executing an SQL statement | EXPLAIN                                        |
| Calling a table stored procedure                                       | CALL STATEMENT                                 |

## View-Related Syntax

| Purpose | Syntax |
|---------|--------|
|---------|--------|

|                                         |                              |
|-----------------------------------------|------------------------------|
| Creating a view from a SELECT result.   | CREATE VIEW AS               |
| Querying views in a database.           | SHOW VIEWS                   |
| Viewing view column information.        | DESCRIBE VIEW                |
| Displaying the view creation statement. | SHOW CREATE VIEW             |
| Viewing view column information.        | SHOW COLUMNS IN VIEW         |
| Renaming a view.                        | ALTER VIEW RENAME TO         |
| Modifying view properties.              | ALTER VIEW SET TBLPROPERTIES |
| Deleting a View                         | DROP VIEW                    |

## Function-Related Syntax

| Purpose                                    |                 |
|--------------------------------------------|-----------------|
| Creating a Function                        | CREATE FUNCTION |
| Viewing the Syntax for Creating a Function | SHOW FUNCTION   |
| Deleting a Function                        | DROP FUNCTION   |

## DML Syntax

| Purpose                                                                                     |                  |
|---------------------------------------------------------------------------------------------|------------------|
| Inserting a row of data                                                                     | INSERT STATEMENT |
| Replacing a row of data                                                                     | INSERT OVERWRITE |
| Row-level data update operation that can be used to replace the INSERT OVERWRITE operation. | MERGE INTO       |
| Querying Iceberg table metadata                                                             | TABLE METADATA   |

|                                           |                  |
|-------------------------------------------|------------------|
| Inserting query results into a data table | INSERT INTO      |
| Deleting data from an Iceberg table       | DELETE STATEMENT |
| Updating specified rows                   | UPDATE           |

DQL Syntax

|            |                  |
|------------|------------------|
| Purpose    |                  |
| Data query | SELECT STATEMENT |

# Overview of Standard Presto Syntax

Last updated: 2026-05-21 14:37:37

The kernel of the standard Presto engine is self-developed based on Presto 0.242. It is compatible with the native Presto syntax and behavior and applies to interactive query and analysis. For detailed syntax, see the Presto syntax documentation.

 **Note:**  
The Presto engine is deprecated and only available for existing users.

| Purpose                            | Syntax          | Whether Supported | Remarks                                                                                                                                                     |
|------------------------------------|-----------------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modifying a Function Definition    | ALTER FUNCTION  | No                |                                                                                                                                                             |
| Renaming a Schema                  | ALTER SCHEMA    | Yes               |                                                                                                                                                             |
| Modifying a Data Table             | ALTER TABLE     | Yes               | Only supports iceberg tables. Specify the target table's Iceberg catalog using the three-part name during execution, or first execute `use iceberg.dbname`. |
| Analyzing a data table             | ANALYZE         | Yes               | Only supports Hive tables.                                                                                                                                  |
| Committing the Current Transaction | COMMIT          | No                |                                                                                                                                                             |
| Creating a Function                | CREATE FUNCTION | No                |                                                                                                                                                             |
| Creates a role.                    | CREATE ROLE     | No                |                                                                                                                                                             |
| Creating a SCHEMA                  | CREATE SCHEMA   | No                |                                                                                                                                                             |

|                                                                        |                    |     |  |
|------------------------------------------------------------------------|--------------------|-----|--|
| Creating Data Table                                                    | CREATE TABLE       | No  |  |
| Creating a Table Using SELECT                                          | CREATE TABLE AS    | No  |  |
| Creating a View                                                        | CREATE VIEW        | No  |  |
| Deallocating a PREPARE Statement                                       | DEALLOCATE PREPARE | No  |  |
| Delete the data.                                                       | DELETE             | No  |  |
| Displaying column information                                          | DESCRIBE           | Yes |  |
| Displaying PREPARE Input Information                                   | DESCRIBE INPUT     | No  |  |
| Displaying PREPARE Output Information                                  | DESCRIBE OUTPUT    | No  |  |
| Deleting a Function                                                    | DROP FUNCTION      | No  |  |
| Deletes roles.                                                         | DROP ROLE          | No  |  |
| Dropping a SCHEMA                                                      | DROP SCHEMA        | No  |  |
| Deleting a data table                                                  | DROP TABLE         | No  |  |
| Deleting a View                                                        | DROP VIEW          | No  |  |
| Executing a PREPARE Statement                                          | EXECUTE            | Yes |  |
| Displaying the logical or physical plan for executing an SQL statement | EXPLAIN            | Yes |  |

|                                                              |                    |     |                                                                                                                                                                 |
|--------------------------------------------------------------|--------------------|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Executing an SQL Statement and Displaying Its Execution Plan | EXPLAIN<br>ANALYZE | Yes |                                                                                                                                                                 |
| Authorizing                                                  | GRANT              | No  |                                                                                                                                                                 |
| Granting a Role to a Specified Object                        | GRANT ROLE         | No  |                                                                                                                                                                 |
| Inserts data.                                                | INSERT             | Yes | If the table is an Iceberg table, specify the target table's Iceberg catalog using the three-part name during execution, or first execute `use iceberg.dbname`. |
| Creating a PREPARE Statement                                 | PREPARE            | Yes |                                                                                                                                                                 |
| Resetting a Specified SESSION to Its Default Value           | RESET<br>SESSION   | Yes |                                                                                                                                                                 |
| Cancel authorization.                                        | REVOKE             | No  |                                                                                                                                                                 |
| Revoking Granted Roles                                       | REVOKE<br>ROLES    | No  |                                                                                                                                                                 |
| Roll back transaction                                        | ROLLBACK           | No  |                                                                                                                                                                 |
| Querying data                                                | SELECT             | Yes |                                                                                                                                                                 |
| Setting a Role                                               | SET ROLE           | No  |                                                                                                                                                                 |
| Setting the Value of a Specified SESSION                     | SET SESSION        | Yes |                                                                                                                                                                 |
| Displaying the CATALOG List                                  | SHOW<br>CATALOGS   | Yes |                                                                                                                                                                 |
| Displaying Column Information of a                           | SHOW<br>COLUMNS    | Yes |                                                                                                                                                                 |



| Table                                      |                      |     |  |
|--------------------------------------------|----------------------|-----|--|
| Displaying Function Information            | SHOW CREATE FUNCTION | No  |  |
| Displaying Table Creation Information      | SHOW CREATE TABLE    | Yes |  |
| Displaying View Creation Information       | SHOW CREATE VIEW     | No  |  |
| Displaying the Function List               | SHOW FUNCTIONS       | Yes |  |
| Displaying Permissions of a Specified User | SHOW GRANTS          | No  |  |
| Displaying the Granted Role List           | SHOW ROLE GRANTS     | No  |  |
| Displaying the Role List                   | SHOW ROLES           | No  |  |
| Displaying the SCHEMA List                 | SHOW SCHEMAS         | Yes |  |
| Displaying the SESSION List                | SHOW SESSION         | Yes |  |
| Displaying Table Statistics                | SHOW STATS           | Yes |  |
| Displaying the Table List                  | SHOW TABLES          | Yes |  |
| Starting a Transaction                     | START TRANSACTION    | No  |  |
| Truncating a Table                         | TRUNCATE             | No  |  |
| Updating Table Content                     | UPDATE               | No  |  |

|                                           |        |     |  |
|-------------------------------------------|--------|-----|--|
| Specifying the Default SCHEMA or Database | USE    | Yes |  |
| Defining an Inline Table                  | VALUES | Yes |  |

# SELECT

## Description

Query data.

## Syntax

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expr [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [, ...] ]
[ OFFSET count [ { ROW | ROWS } ] ]
[ { LIMIT [ count | ALL ] } ]
```

from\_item:

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
```

```
from_item join_type from_item [ ON join_condition | USING ( join_column
[, ...] ) ]
```

join\_type:

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
```

```
FULL [ OUTER ] JOIN  
CROSS JOIN
```

grouping\_element:

```
( )  
expression  
GROUPING SETS ( ( column [, ...] ) [, ...] )  
CUBE ( column [, ...] )  
ROLLUP ( column [, ...] )
```

## Examples

### WITH Clause

The WITH clause defines named relationships used in a query. It allows you to flatten nested queries or simplify subqueries. For example, the following queries are equivalent:

```
SELECT a, b  
FROM (  
    SELECT a, MAX(b) AS b FROM t GROUP BY a  
) AS x;  
  
WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)  
SELECT a, b FROM x;
```

It also applies to multiple subqueries:

```
WITH  
    t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),  
    t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)  
SELECT t1.*, t2.*  
FROM t1  
JOIN t2 ON t1.a = t2.a;
```

In addition, the relationships defined in a WITH clause can be joined:

```
WITH  
    x AS (SELECT a FROM t),
```

```
y AS (SELECT a AS b FROM x),  
z AS (SELECT b AS c FROM y)  
SELECT c FROM z;
```

## GROUP BY Clause

A GROUP BY clause groups the SELECT output by the specified fields.

The following queries are equivalent. They are both grouped by nationkey. The first query is grouped by the column ordinal, which is the second column, while the second query is grouped directly by the column name:

```
SELECT count(*), nationkey FROM customer GROUP BY 2;  
  
SELECT count(*), nationkey FROM customer GROUP BY nationkey;
```

You can also omit the grouping fields in the query:

```
SELECT count(*) FROM customer GROUP BY mktsegment;
```

| _col0 |
|-------|
| 29968 |
| 30142 |
| 30189 |
| 29949 |
| 29752 |

(5 rows)

## Complex Grouping Operations

Presto also supports complex aggregation operations using the GROUPING SETS, CUBE, and ROLLUP syntax. This syntax allows users to aggregate multiple column sets within a single query.

Complex grouping operations are usually equivalent to a UNION ALL of simple GROUP BY expressions, as shown in the following example.

### GROUPING SETS

GROUPING SETS allows users to specify multiple lists of columns for grouping. Columns that are not part of a given sublist of grouping columns are set to NULL.

```
SELECT * FROM shipping;
```

```

  origin_state | origin_zip | destination_state | destination_zip |
package_weight
-----+-----+-----+-----+-----
-----
California    |    94131   | New Jersey       |    8648   |
13
California    |    94131   | New Jersey       |    8540   |
42
New Jersey    |    7081    | Connecticut      |    6708   |
225
California    |    90210   | Connecticut      |    6927   |
1337
California    |    94131   | Colorado         |    80302  |
5
New York      |    10002   | New Jersey       |    8540   |
3
(6 rows)

```

The semantics of **GROUPING SETS** can be demonstrated with the following example query:

```

SELECT origin_state, origin_zip, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));

```

```

  origin_state | origin_zip | destination_state | _col0
-----+-----+-----+-----
New Jersey    | NULL      | NULL             |    225
California    | NULL      | NULL             |   1397
New York      | NULL      | NULL             |     3
California    |    90210  | NULL             |   1337
California    |    94131  | NULL             |     60
New Jersey    |    7081   | NULL             |    225
New York      |    10002  | NULL             |     3
NULL          | NULL      | Colorado         |     5

```

```
NULL      | NULL      | New Jersey      |      58
NULL      | NULL      | Connecticut     |     1562
(10 rows)
```

Logically, the preceding query can be considered equivalent to a **UNION ALL** of multiple **GROUP BY** queries:

```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state

UNION ALL

SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip

UNION ALL

SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

## CUBE

The **CUBE** operator generates all possible grouping sets (that is, the power set) for a given set of columns. It allows you to aggregate various combinations of columns. For example, consider the following query:

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY CUBE (origin_state, destination_state);

-- Equivalent to:
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

```

origin_state | destination_state | _col0
-----+-----+-----
California   | New Jersey        | 55
California   | Colorado          | 5
New York     | New Jersey        | 3
New Jersey   | Connecticut       | 225
California   | Connecticut       | 1337
California   | NULL              | 1397
New York     | NULL              | 3
New Jersey   | NULL              | 225
NULL         | New Jersey        | 58
NULL         | Connecticut       | 1562
NULL         | Colorado          | 5
NULL         | NULL              | 1625
(12 rows)

```

## ROLLUP

The ROLLUP operator generates all possible subtotals for a given set of columns. For example, consider the following query:

```

SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ROLLUP (origin_state, origin_zip);

```

```

origin_state | origin_zip | _col2
-----+-----+-----
California   | 94131     | 60
California   | 90210     | 1337
New Jersey   | 7081      | 225
New York     | 10002     | 3
California   | NULL      | 1397
New York     | NULL      | 3
New Jersey   | NULL      | 225
NULL         | NULL      | 1625
(8 rows)

```

-- Equivalent to:

```

SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping

```

```
GROUP BY GROUPING SETS ((origin_state, origin_zip), (origin_state), ());
```

## Combining multiple grouping expressions

Using multiple grouping expressions in a single query is interpreted as having cross-product semantics. For example, the following query:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    ROLLUP (origin_zip);
```

-- Can be rewritten as:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    GROUPING SETS ((origin_zip), ());
```

-- Logically equivalent to:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, destination_state));
```

| origin_state | destination_state | origin_zip | _col3 |
|--------------|-------------------|------------|-------|
| New York     | New Jersey        | 10002      | 3     |
| California   | New Jersey        | 94131      | 55    |
| New Jersey   | Connecticut       | 7081       | 225   |
| California   | Connecticut       | 90210      | 1337  |
| California   | Colorado          | 94131      | 5     |
| New York     | New Jersey        | NULL       | 3     |
| New Jersey   | Connecticut       | NULL       | 225   |
| California   | Colorado          | NULL       | 5     |
| California   | Connecticut       | NULL       | 1337  |
| California   | New Jersey        | NULL       | 55    |



```
(10 rows)
```

The ALL and DISTINCT quantifiers determine whether duplicate grouping sets produce distinct output rows. This is particularly useful when multiple complex grouping sets are combined within a single query. For example, the following query:

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ALL
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

```
-- Equivalent to:
```

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

```
-- However, if a query uses the DISTINCT quantifier in the GROUP BY
clause:
```

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY DISTINCT
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

```
-- Only a unique grouping set is generated:
```

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight)
```

```
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

## GROUPING Operation

`grouping(col1, ..., colN) -> bigint` The grouping operation returns a bit set converted to decimal, indicating which columns are present in the grouping. It must be used with **GROUPING SETS**, **ROLLUP**, **CUBE**, or **GROUP BY**, and its arguments must exactly match the columns referenced in the corresponding **GROUPING SETS**, **ROLLUP**, **CUBE**, or **GROUP BY** clause.

To compute the result bit set for a specific row, bits are assigned to parameter columns, with the rightmost column being the least significant bit. For a given grouping, if the corresponding column is included in the grouping, the bit is set to 0; otherwise, it is set to 1. For example, consider the following query:

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight),
       grouping(origin_state, origin_zip, destination_state)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));
```

| origin_state | origin_zip | destination_state | _col3 | _col4 |
|--------------|------------|-------------------|-------|-------|
| California   | NULL       | NULL              | 1397  | 3     |
| New Jersey   | NULL       | NULL              | 225   | 3     |
| New York     | NULL       | NULL              | 3     | 3     |
| California   | 94131      | NULL              | 60    | 1     |
| New Jersey   | 7081       | NULL              | 225   | 1     |
| California   | 90210      | NULL              | 1337  | 1     |
| New York     | 10002      | NULL              | 3     | 1     |
| NULL         | NULL       | New Jersey        | 58    | 6     |
| NULL         | NULL       | Connecticut       | 1562  | 6     |

```
NULL      | NULL      | Colorado      | 5 | 6
(10 rows)
```

```
-- In the result above, the first grouping includes only the
origin_state column, excluding the origin_zip and destination_state
columns. The bit set constructed for this grouping is 011, where the
most significant bit represents origin_state.
```

## HAVING Clause

The HAVING clause is used with aggregate functions and the GROUP BY clause to control which groups are selected. The HAVING clause eliminates groups that do not meet the given condition. HAVING filters groups after the grouping and aggregation are computed.

The following example queries the customer table and selects groups where the account balance is greater than the specified value:

```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
```

```
_col0 | mktsegment | nationkey | totalbal
-----+-----+-----+-----
1272 | AUTOMOBILE | 19 | 5856939
1253 | FURNITURE | 14 | 5794887
1248 | FURNITURE | 9 | 5784628
1243 | FURNITURE | 12 | 5757371
1231 | HOUSEHOLD | 3 | 5753216
1251 | MACHINERY | 2 | 5719140
1247 | FURNITURE | 8 | 5701952
(7 rows)
```

## UNION | INTERSECT | EXCEPT Clause

UNION, INTERSECT, and EXCEPT are all set operations. These clauses are used to combine the results of multiple SELECT statements into a single result set:

```
query UNION [ALL | DISTINCT] query
```

```
query INTERSECT [DISTINCT] query
query EXCEPT [DISTINCT] query
```

The ALL or DISTINCT parameter determines which rows are included in the final result set. If the ALL parameter is specified, all rows are included even if they are identical. If the DISTINCT parameter is specified, only unique rows are included in the combined result set. If no parameter is specified, the default behavior is DISTINCT. The ALL parameter is not supported for INTERSECT or EXCEPT.

Multiple set operations are processed from left to right unless the order is explicitly specified using parentheses. Additionally, INTERSECT has higher precedence than EXCEPT and UNION. This means that A UNION B INTERSECT C EXCEPT D is the same as A UNION (B INTERSECT C) EXCEPT D.

## UNION

UNION combines all rows from the result set of the first query with the rows from the result set of the second query. The following is a simple example of a UNION clause. It selects the value 13 and combines this result set with the result set of the second query, which selects the value 42:

```
SELECT 13
UNION
SELECT 42;

 _col0
-----
    13
    42
(2 rows)
```

The following query demonstrates the difference between UNION and UNION ALL. It selects the value 13 and combines this result set with the result set of the second query, which selects the values 42 and 13:

```
SELECT 13
UNION
SELECT * FROM (VALUES 42, 13);

 _col0
-----
```

```

13
42
(2 rows)

-----

SELECT 13
UNION ALL
SELECT * FROM (VALUES 42, 13);

_col0
-----
13
42
13
(2 rows)

```

## INTERSECT

INTERSECT returns only the rows that are present in both the result set of the first query and the result set of the second query. The following is a simple example of an INTERSECT clause. It selects the values 13 and 42 and combines this result set with the result set of the second query, which selects the value 13. Since 42 exists only in the result set of the first query, it is not included in the final result:

```

SELECT * FROM (VALUES 13, 42)
INTERSECT
SELECT 13;

_col0
-----
13
(2 rows)

```

## EXCEPT

EXCEPT returns rows that are present in the result set of the first query but not in the result set of the second query. The following is a simple example of an EXCEPT clause. It selects the values 13 and 42 and combines this result set with the result set of the second query, which selects the value 13. Since 13 also exists in the result set of the second query, it is not included in the final result:

```
SELECT * FROM (VALUES 13, 42)
EXCEPT
SELECT 13;

 _col0
-----
    42
(2 rows)
```

## ORDER BY Clause

The ORDER BY clause is used to sort the result set by one or more output expressions. Each expression can consist of an output column or be an ordinal number that selects an output column by its position (starting from 1). The ORDER BY clause is evaluated after any GROUP BY or HAVING clause and before any OFFSET, LIMIT, or FETCH FIRST clause. The default null ordering is NULLS LAST, regardless of the sort direction.

```
ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...]
```

## OFFSET Clause

The OFFSET clause is used to discard a specified number of leading rows from the result set.

```
OFFSET count [ ROW | ROWS ]
```

If an ORDER BY clause is present, the OFFSET clause is evaluated on the sorted result set, and the result set remains sorted after the leading rows are discarded.

```
SELECT name FROM nation ORDER BY name OFFSET 22;

      name
-----
UNITED KINGDOM
UNITED STATES
VIETNAM
(3 rows)
```

Otherwise, which rows are discarded is arbitrary. If the count specified in the OFFSET clause equals or exceeds the size of the result set, the final result will be empty.

## LIMIT Clause

The LIMIT clause limits the number of rows in the result set. LIMIT ALL is equivalent to omitting the LIMIT clause, indicating that no row limit is applied.

```
LIMIT { count | ALL }
```

The following example queries a large table. However, because it lacks an ORDER BY clause, the LIMIT clause will restrict the output to only five rows.

```
SELECT orderdate FROM orders LIMIT 5;
```

```
o_orderdate
-----
1996-04-14
1992-01-15
1995-02-01
1995-11-12
1992-04-26
(5 rows)
```

LIMIT ALL is equivalent to omitting the LIMIT clause. If an OFFSET clause is present, the LIMIT clause is evaluated after the OFFSET clause.

```
SELECT * FROM (VALUES 5, 2, 4, 1, 3) t(x) ORDER BY x OFFSET 2 LIMIT 2;
```

```
x
--
3
4
(2 rows)
```

## TABLESAMPLE

### BERNOULLI:

The probability of a row being selected into the table sample is determined by the sampling percentage. When a table is sampled using the Bernoulli method, all physical blocks of the table are scanned. A runtime-computed random value is compared with the sampling percentage to decide whether to skip certain rows. The probability of whether a row is

included in the result is independent of other rows. This does not reduce the time required to read the sampled table from disk. If the sampling output is processed further, it may affect the total query time.

## SYSTEM:

This sampling method divides a table into logical data segments and samples the table at this granularity. The method either selects all rows in a specific data segment or skips them, based on a comparison between a runtime-computed random value and the sampling percentage. The rows selected in a system sample depend on the connector used. For example, when the sampling method is used with Hive, it depends on how the data is laid out on HDFS. This method does not guarantee independent sampling probabilities.

```
SELECT *
FROM users TABLESAMPLE BERNOULLI (50);

SELECT *
FROM users TABLESAMPLE SYSTEM (75);

SELECT o.*, i.*
FROM orders o TABLESAMPLE SYSTEM (10)
JOIN lineitem i TABLESAMPLE BERNOULLI (40)
  ON o.orderkey = i.orderkey;
```

## UNNEST

UNNEST can be used to expand an array (ARRAY) or a map (MAP) into a relation. An array expands into a single column, while a map expands into two columns (key, value). UNNEST can also be used with multiple arguments. In this case, they expand into multiple columns, with the number of rows matching the highest cardinality argument (other columns are padded with nulls). UNNEST can optionally use a WITH ORDINALITY clause, which adds an extra ordinality column at the end. UNNEST is typically used with JOIN and can reference columns from the relation on the left side of the join.

```
-- Using a single array column
SELECT student, score
FROM tests
CROSS JOIN UNNEST(scores) AS t (score);

-- Using multiple array columns
```



```

SELECT numbers, animals, n, a
FROM (
  VALUES
    (ARRAY[2, 5], ARRAY['dog', 'cat', 'bird']),
    (ARRAY[7, 8, 9], ARRAY['cow', 'pig'])
) AS x (numbers, animals)
CROSS JOIN UNNEST(numbers, animals) AS t (n, a);

```

| numbers   | animals          | n    | a    |
|-----------|------------------|------|------|
| [2, 5]    | [dog, cat, bird] | 2    | dog  |
| [2, 5]    | [dog, cat, bird] | 5    | cat  |
| [2, 5]    | [dog, cat, bird] | NULL | bird |
| [7, 8, 9] | [cow, pig]       | 7    | cow  |
| [7, 8, 9] | [cow, pig]       | 8    | pig  |
| [7, 8, 9] | [cow, pig]       | 9    | NULL |

(6 rows)

```
-- WITH ORDINALITY clause:
```

```

SELECT numbers, n, a
FROM (
  VALUES
    (ARRAY[2, 5]),
    (ARRAY[7, 8, 9])
) AS x (numbers)
CROSS JOIN UNNEST(numbers) WITH ORDINALITY AS t (n, a);

```

| numbers   | n | a |
|-----------|---|---|
| [2, 5]    | 2 | 1 |
| [2, 5]    | 5 | 2 |
| [7, 8, 9] | 7 | 1 |
| [7, 8, 9] | 8 | 2 |
| [7, 8, 9] | 9 | 3 |

(5 rows)

```
-- Using a single map (MAP) column:
```

```

SELECT
  animals, a, n
FROM (

```

```
VALUES
  (MAP (ARRAY['dog', 'cat', 'bird'], ARRAY[1, 2, 0])),
  (MAP (ARRAY['dog', 'cat'], ARRAY[4, 5]))
) AS x (animals)
CROSS JOIN UNNEST(animals) AS t (a, n);
```

| animals                    | a    | n |
|----------------------------|------|---|
| {"cat":2,"bird":0,"dog":1} | dog  | 1 |
| {"cat":2,"bird":0,"dog":1} | cat  | 2 |
| {"cat":2,"bird":0,"dog":1} | bird | 0 |
| {"cat":5,"dog":4}          | dog  | 4 |
| {"cat":5,"dog":4}          | cat  | 5 |

(5 rows)

## Joins

Joins allow you to combine data from multiple relations.

### CROSS JOIN

A cross join returns the Cartesian product (all combinations) of two relations. You can specify a cross join by using the explicit CROSS JOIN syntax or by specifying multiple relations in the FROM clause.

The following two queries are equivalent:

```
SELECT *
FROM nation
CROSS JOIN region;

SELECT *
FROM nation, region;
```

The nation table contains 25 rows, and the region table contains 5 rows. Therefore, a cross join between the two tables will produce 125 rows:

```
SELECT n.name AS nation, r.name AS region
FROM nation AS n
CROSS JOIN region AS r
ORDER BY 1, 2;
```

```
      nation      |      region
-----+-----
ALGERIA          | AFRICA
ALGERIA          | AMERICA
ALGERIA          | ASIA
ALGERIA          | EUROPE
ALGERIA          | MIDDLE EAST
ARGENTINA        | AFRICA
ARGENTINA        | AMERICA
...
(125 rows)
```

## Qualifying Column Names

When the two relations in a join have columns with the same name, you must qualify the column references using the relation alias (if the relation has one) or the relation name:

```
SELECT nation.name, region.name
FROM nation
CROSS JOIN region;

SELECT n.name, r.name
FROM nation AS n
CROSS JOIN region AS r;

SELECT n.name, r.name
FROM nation n
CROSS JOIN region r;
```

The following query will fail and display the error message "Column 'name' is ambiguous":

```
SELECT nameFROM nationCROSS JOIN region;
```

## USING

You can write shorter queries by using the **USING** clause when the two tables being joined have the same join key name.

For example:

```

SELECT *
FROM table_1
JOIN table_2
ON table_1.key_A = table_2.key_A AND table_1.key_B = table_2.key_B

-- Can be rewritten as:
SELECT *
FROM table_1
JOIN table_2
USING (key_A, key_B)

```

The output of a join using USING includes the join key columns (key\_A and key\_B in the example above), followed by the remaining columns from table\_1, and finally the remaining columns from table\_2. Note that the join keys are not included in the original table lists for referencing them in queries. You cannot access them using table prefixes. If you run `SELECT table_1.*, table_2.*`, the join columns will not be included in the output.

The following two queries are equivalent:

```

SELECT *
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)
LEFT JOIN (
    VALUES
        (1, 3, 100),
        (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
USING (key_A, key_B)

-----

SELECT key_A, key_B, table_1.*, table_2.*
FROM (
    VALUES
        (1, 3, 10),
        (2, 4, 20)
) AS table_1 (key_A, key_B, y1)

```

```

LEFT JOIN (
  VALUES
    (1, 3, 100),
    (2, 4, 200)
) AS table_2 (key_A, key_B, y2)
USING (key_A, key_B)

```

```

key_A | key_B | y1 | y2
-----+-----+----+----
      1 |      3 | 10 | 100
      2 |      4 | 20 | 200
(2 rows)

```

## Subquery

A subquery is an expression that consists of a query. When a subquery references columns from outside the subquery, it is called a correlated subquery.

## EXISTS

EXISTS is used to determine whether a subquery returns any rows:

```

SELECT name
FROM nation
WHERE EXISTS (SELECT * FROM region WHERE region.regionkey =
nation.regionkey)

```

## IN

It is used to determine whether any value generated by a subquery equals the provided expression. The subquery must generate exactly one column:

```

SELECT name
FROM nation
WHERE regionkey IN (SELECT regionkey FROM region)

```

## Scalar Subqueries

A scalar subquery is a non-correlated subquery that returns zero or one row. If the subquery produces more than one row, an error will be raised. If the subquery returns no rows, the returned value will be NULL.

```
SELECT name
FROM nation
WHERE regionkey = (SELECT max(regionkey) FROM region)
```

# Reserved Words

Last updated: 2026-05-21 14:59:34

## Notes

If any of these identifiers are reserved words used as identifiers for database names, table names, column names, function names, view names, and so on, you must add the default escape character:

```
Identifier: ` backtick
```

```
For example: hour -> `hour`
```

```
create table `hour` (  
  id string,  
  `asc` int  
)
```

If you need to create a `column` with a name that is a keyword, such as `hour` or `asc`, you must enclose the `column` name in backticks.

## Key Reserved Words

### A

- ALL
- ALTER
- AND
- ANY
- AS
- AUTHORIZATION

### B

- BETWEEN
- BOTH
- BY

### C

- CALL
- CASE
- CAST
- CHECK
- CLUSTER
- COLLATE
- COLUMN
- CONSTRAINT
- CREATE
- CROSS
- CUBE
- CURRENT\_DATE
- CURRENT\_TIME
- CURRENT\_TIMESTAMP
- CURRENT\_USER
- CURSOR

## D

- DEALLOCATE
- DEFAULT
- DELETE
- DESCRIBE
- DISTINCT
- DISTRIBUTE
- DROP

## E

- ELSE
- END
- ESCAPE
- EXCEPT
- EXECUTE
- EXISTS
- EXPLAIN
- EXTRACT



## F

- FETCH
- FILTER
- FOR
- FOREIGN
- FROM
- FULL
- FALSE

## G

- GRANT
- GROUP
- GROUPING

## H

- HAVING

## I

- IN
- INNER
- INSERT
- INTERSECT
- INTERVAL
- INTO
- IS

## J

- JOIN

## L

- LATERAL
- LEADING
- LEFT
- LIKE
- LIMIT

- LOCALTIME
- LOCALTIMESTAMP

## M

- MERGE
- MINUS

## N

- NATURAL
- NEW
- NEXT
- NORMALIZE
- NOT
- NULL

## O

- OFFSET
- ON
- ONLY
- OR
- ORDER
- OUTER
- OVER
- OVERLAPS

## P

- PARTITION
- PATTERN
- PERCENTILE\_CONT
- PERCENTILE\_DISC
- PERMUTE
- PREPARE
- PRIMARY

## R

- RANGE

- RECURSIVE
- REFERENCES
- RIGHT
- ROLLUP
- ROW
- ROWS

## S

- SELECT
- SEMI
- SESSION\_USER
- SET
- SOME

## T

- TABLE
- THEN
- TIME
- TO
- TRAILING
- TRUE

## U

- UESCAPE
- UNION
- UNIQUE
- UNKNOWN
- UNNEST
- UPDATE
- USER
- USING

## V

- VALUES

## W

- WHEN
- WHERE
- WINDOW
- WITH
- WITHIN