Data Lake Compute

SQL構文

製品ドキュメント

著作権声明

商標に関する声明

サービス声明

本ドキュメントは、お客様にTencent Cloudの全部また一部の製品・サービスの概要をご紹介することを目的としておりますが、一部の製品・サービス内容は変更される可能性があります。お客様がご購入されるTencent Cloud製品・サービスの種類やサービス基準などは、お客様とTencent Cloudとの間の締結された商業契約に基づきます。別段の合意がない限り、Tencent Cloudは本ドキュメントの内容に関して、明示または黙示の一切保証もしません。

カタログ:

SQL構文
SuperSQL構文
SuperSQL 语法概览
統一構文
常用数据类型
DDL構文
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 DROP COLUMN
ALTER TABLE ADD PARTATION
SHOW PARTITIONS
ALTER TABLE DROP PARTITION
ALTER TABLE ADD PARTITION FIELD
ALTER TABLE DROP PARTITION FIELD
ALTER TABLE ... RENAME COLUMN
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構文
INSERT STATEMENT
INSERT INTO
INSERT OVERWRITE
MERGE INTO
UPDATE
DELETE STATEMENT
TABLE METADATA
DQL構文
SELECT STATEMENT
「Iceberg テーブル構文」
DDL 構文
DML構文
DQL構文
Procedure
Iceberg外部テーブルとネイティブテーブルの構文の違い
マテリアライズドビュー構文
SQL暗黙の型変換
関数
統一関数

統一関数の概要
二項関数
ビット演算関数
集約関数
日時関数
JSON 関数
数学関数
文字列関数
集約関数
ウィンドウ関数
他の関数
Presto 組み込み関数
Hive 関数対応表
標準 Spark 構文概要
標準 Presto 構文の概要
予約語

SQL構文
SuperSQL構文
SuperSQL 语法概览

最終更新日：：2025-12-25 11:51:48

DLCは、標準SQL構文のセットを使用することで、DLC Serverless SparkとDLC Serverless Prestoエンジン上でほぼシームレスに実行できます。メタデータと分析構文、関数は、基本的にHiveおよびSpark構文と互換性があり、カスタム関数をサポートしています。
システム組み込み関数のサポート範囲については、統一関数概要を参照してください。Presto組み込み関数を使用する場合の使用方法と関数サポート範囲については、Presto組み込み関数を参照してください。
データレイクコンピューティングで外部Icebergテーブルに対してデータクエリと分析を行う場合、一部の構文がネイティブテーブルと異なる場合があります。詳細については、Iceberg外部テーブルとネイティブテーブルの構文の違いを参照してください。
DLCがサポートする構文は以下の表の通りです：

DDL構文

データベース関連構文

用途	構文
新規データベース	CREATE DATABASE
そのメタデータで定義されているすべてのデータベースを表示します	SHOW DATABASES
データベース属性を表示	DESCRIBE DATABASE
データベース属性の変更	ALTER DATABASE SET DBPROPERTIES
データベースの保存場所変更	ALTER DATABASE SET LOCATION
データベースを削除	DROP DATABASE

データテーブル関連構文

用途	構文
新規データテーブル	CREATE TABLE
テーブルスナップショットを更新	REPLACE TABLE AS SELECT

データテーブルの作成情報を照会	SHOW CREATE TABLE
テーブル属性を照会	SHOW TBLPROPERTIES
データベース内のすべてのテーブルを照会	SHOW TABLES
データテーブルの列情報及びメタデータ情報を表示	DESCRIBE TABLE
データテーブルの列情報を照会	SHOW COLUMNS IN TABLE
データテーブルに列を追加	ALTER TABLE ADD COLUMNS
データテーブルに列を追加	ALTER TABLE ADD COLUMN AFTER/FIRST
変更前の名称	ALTER TABLE ... RENAME COLUMN
データテーブルの特定のフィールドを削除	ALTER TABLE DROP COLUMN
データテーブルにパーティション情報を追加	ALTER TABLE ADD PARTATION
テーブルのパーティションを一覧表示	SHOW PARTITIONS
データテーブルのパーティション情報を削除	ALTER TABLE DROP PARTITION
Iceberg テーブルにパーティションフィールドを追加	ALTER TABLE ADD PARTITION FIELD
Iceberg テーブルからパーティションフィールドを削除	ALTER TABLE DROP PARTITION FIELD
データテーブルの属性変更	ALTER TABLE SET TBLPROPERTIES
データテーブルの保存場所変更	ALTER TABLE SET LOCATION
データテーブルの並べ替え方法を変更	ALTER TABLE ... WRITE ORDERED BY
パーティションテーブルの割り当て戦略を変更	ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION
識別子フィールド属性を追加	ALTER TABLE ... SET IDENTIFIER FIELDS
identifier fields 属性を削除	ALTER TABLE ... DROP IDENTIFIER FIELDS

パーティション情報を更新	MSCK REPAIR TABLE
データテーブルを統計	ANALYZE TABLES
メタデータテーブルを削除	DROP TABLE
sqlの論理または物理プランを表示する	EXPLAIN
テーブルストアドプロシージャを呼び出す	CALL STATEMENT

ビュー関連の構文

用途	構文
select結果をビューとして作成する	CREATE VIEW AS
データベース内のビューをクエリする	SHOW VIEWS
ビューの列情報を表示する	DESCRIBE VIEW
ビューの作成ステートメントを表示する	SHOW CREATE VIEW
ビューの列情報を表示する	SHOW COLUMNS IN VIEW
ビューの名前を変更する	ALTER VIEW RENAME TO
ビューの属性を変更する	ALTER VIEW SET TBLPROPERTIES
ビューの削除	DROP VIEW

関数関連の構文

用途	
関数を作成	CREATE FUNCTION
関数作成の構文を確認	SHOW FUNCTION
関数を削除	DROP FUNCTION

DML構文

用途

データを行に挿入	INSERT STATEMENT
データを行に置き換える	INSERT OVERWRITE
行レベルのデータ更新操作は、INSERT OVERWRITE操作の代わりに使用できます	MERGE INTO
Iceberg テーブルメタデータクエリ	TABLE METADATA
クエリ結果をデータテーブルに挿入	INSERT INTO
Iceberg テーブルからデータを削除	DELETE STATEMENT
指定行を更新	UPDATE

DQL 構文

用途	
データクエリ	SELECT STATEMENT

関連クエリの予約語については、予約語を参照してください。

統一構文
常用数据类型

最終更新日：：2025-12-25 11:51:48

データ型の分類	データ型	説明
数字	ByteType：BYTE, TINYINT	1バイトの符号付き整数、数字の範囲は-128から127です
	ShortType：SHORT, SMALLINT	2バイトの符号付き整数、数字の範囲は-32768から3276です
	IntegerType：INT, INTEGER	4バイト符号付き整数、数値の範囲は-2147483648から2147483647まで
	LongType：LONG, BIGINT	8バイトの符号付き整数で、範囲は-9223372036854775808から9223372036854775807までです
	FloatType：FLOAT, REAL	4バイト単精度浮動小数点数
	DoubleType：DOUBLE	8バイト倍精度浮動小数点数
	DecimalType：DECIMAL, DEC, NUMERIC	任意精度の符号付き10進数
文字列	StringType：STRING	文字列の値、例えば：'abc'
バイト	BinaryType：BINARY	バイトシーケンスの値
ブール	BooleanType：BOOLEAN	ブール値、例：true/false
時間	DateType：DATE	年、月、日のフィールド値で構成される値、タイムゾーンなし 例：'2023-10-01'
	TimestampType：TIMESTAMP	年、月、日、時間、分、秒のフィールド値およびローカルタイムゾーンで構成される値 例：'2023-10-01 23:59:59'
複合型	ArrayType：ARRAY<element_type>	element_type タイプの要素で構成されるシーケンス値 例えば：array<int>

	MapType：MAP<key_type, value_type>	key_type でキーのタイプを指定し、value_type で値のタイプを指定する、キーと値のペアで構成される値 例えば：map<string, int>
	StructType：STRUCT<field1_name: field1_type, field2_name: field2_type, …>	fields で構成される構造値 例えば：struct<id:int, grade:string>

DDL構文
CREATE DATABASE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：データベースを作成するほか、テーブルまたはパーティションテーブルの保存場所を指定して変更することもできます。

構文

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

パラメータ

- `DATABASE|SCHEMA`：同じ意味で、どちらも使用できます。
- `database_name`：データベース名。
- `database_comment`：データベースコメント。
- `[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]`：`key-value` の形式でデータベースパラメータを設定します。

例

Create database `db` only if database with same name doesn't exist.

```
CREATE DATABASE IF NOT EXISTS db;
```

Create database `db` only if database with same name doesn't exist with `Comment` and `Database Properties`.

```
CREATE DATABASE db
COMMENT 'db1_name'
WITH DBPROPERTIES('k1' = 'v1','k2' = 'v2');
```

SHOW DATABASES

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：このメタデータで定義されているすべてのデータベースをリストします。DATABASESまたはSCHEMASを使用して同じクエリを実行できます。

構文

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

パラメータ

- `[IN catalog_name]`：データソース名。
- `[LIKE 'regular_expression']`：一致するデータベース名をフィルタリングします。

例

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

DESCRIBE DATABASE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：データベースのプロパティを表示します。

標準構文

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

パラメータ

- `SCHEMA|DATABASE`：指定するライブラリはSCHEMAまたはDATABASEです。
- `EXTENDED`：このライブラリはEXTENDEDであるかどうか。

例

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

ALTER DATABASE
ALTER DATABASE SET DBPROPERTIES

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：データベースに1つ以上の属性を追加し、同じ属性がある場合は上書きされます。

標準構文

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

パラメータ

`database_name`：データベース名。

例

```
-- Alters the database to set properties `author`.
ALTER DATABASE product SET DBPROPERTIES ('author' = 'allen');
```

ALTER DATABASE SET LOCATION

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto。
- 用途：データベースの保存パスを変更します。

標準構文

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

パラメータ

`[database_name]`：データベース名。

例

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

DROP DATABASE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：指定されたデータベースを削除します。

構文

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

パラメータ

- `DATABASE|SCHEMA`：同じ意味で、どちらも使用できます。
- `database_name`：データベース名。
- `RESTRICT`：データベースにテーブルが含まれている場合、そのデータベースは削除されません。未入力の場合、デフォルトはこのモードです。
- `CASCADE`：すべてのデータベーステーブルを強制的に削除します。

例

```
-- Drop the database and it's tables
DROP DATABASE test CASCADE;

-- Drop the database using IF EXISTS
DROP DATABASE IF EXISTS test;
```

CREATE TABLE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：テーブルを作成すると同時にいくつかの属性を設定し、CREATE TABLE AS構文の使用をサポートします。
- 建表ストレージパス：テーブル作成のストレージパスはCOSディレクトリに指定できますが、ファイルには指定できません。

外部テーブル構文

構文

```
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, ... ) ]
```

パラメータ

`USING data_source`：テーブル作成時におけるデータの入力タイプ。現在サポートされている形式：CSV、ORC、PARQUET、ICEBERGなど。
`table_identifier`：テーブル名を指定。三段式（例：catalog.database.table）をサポート。
`COMMENT`：テーブルの説明情報。
`OPTIONS`：USING data_sourceでサポートされる追加パラメータ。保存時のパラメータ注入に使用。
`PARTITIONED BY`：指定された列に基づいてパーティションを作成。
`LOCATION path`：データテーブルの保存パス。
`TBLPROPERTIES`：テーブルのパラメータを指定するための一連のk-v値。

USINGとOPTIONS パラメータの詳細説明

USING CSV

参考リンク： CSVの操作
CSVデータテーブルでサポートされている構成は次のとおりです。

OPTIONS でサポートされているキー	key に対応する value のデフォルト値	意味
sepまたはdelimiter	,	csv保存時の列間の区切り文字、デフォルトは英語のカンマ
mode	PERMISSIVE	データ変換時に期待通りに処理されない場合の処理モードを定義します。 PERMISSIVE：より寛容なモード、デフォルトで、ある行のデータを変換しようと試みます。例えば、ある行に余分な列がある場合、必要な列のみを自動的に取得します。 DROPMALFORMED：期待通りに処理されないデータを破棄します。例えば、ある行に余分な列がある場合、その行は破棄されます。 FAILFAST：csvフォーマットを厳密に要求し、ある行が期待通りでない場合（例えば余分な列がある場合）すぐに失敗します。
encodingまたはcharset	UTF-8	文字列エンコーディングフォーマット。 例えば：UTF-8、US-ASCII、ISO-8859-1、UTF-16BE、UTF-16LE、UTF-16
quote	\"	引用符はシングルクォートかダブルクォートか、エスケープ文字の使用に注意してください
escape	\\	エスケープ文字、エスケープ文字の使用に注意してください
charToEscapeQuoteEscaping	-	引用符内部でエスケープが必要な文字
comment	\u0000	備考情報
header	false	ヘッダーが存在します
inferSchema	false	列のタイプを推測し、推測しない場合は各列が文字列になります

ignoreLeadingWhiteSpace	読み取り：false 書き込み：true	無視する先頭の空文字列
ignoreTrailingWhiteSpace	読み取り：false 書き込み：true	無視する末尾の空文字列
columnNameOfCorruptRecord	_corrupt_record	変換できない列の列名。このパラメータはspark.sql.columnNameOfCorruptRecordの影響を受け、テーブルの設定が優先されます
nullValue	–	nullのストレージ形式。デフォルトは空文字列で、この場合emptyValueの方法で書き込まれます
nanValue	NaN	非数値型の値のストレージ形式
positiveInf	Inf	正の無限大のストレージ形式
negativeInf	-Inf	負の無限大のストレージ形式
compressionまたはcodec	–	圧縮アルゴリズムのクラス名。デフォルトでは圧縮なし。略称を使用可能：bzip2、deflate、gzip、lz4、snappy
timeZone	システムデフォルトのタイムゾーン	デフォルトのタイムゾーン。このパラメータの値はspark.sql.session.timeZoneの影響を受けます（例：Asia/Shanghai）。テーブルの設定が優先されます。
locale	en-US	言語タイプ
dateFormat	yyyy-MM-dd	デフォルトの日付形式
timestampFormat	yyyy-MM-dd'T'HH:mm:ss.SSSXXX	デフォルトの時間形式。LEGACYモード以外ではyyyy-MM-dd'T'HH:mm:ss[.SSS][XXX]となります
multiLine	false	複数行を許可
maxColumns	20480	最大列数
maxCharsPerColumn	-1	各列の最大文字数、-1は制限なしを意味します
escapeQuotes	true	エスケープ引用符
quoteAll	quoteAll	「書き込み時に全文に引用符を追加」
samplingRatio	1.0	サンプリング比率

enforceSchema	true	指定されたスキーマを使用して強制的に読み取り、ヘッダーの定義は無視されます
emptyValue	読み取り 書き込み：\"\"	空値の読み書き形式
lineSep	–	改行文字
inputBufferSize	–	読み取り時のバッファサイズ。このパラメータはspark.sql.csv.parser.inputBufferSizeの影響を受けますが、テーブル設定が優先されます
unescapedQuoteHandling	STOP_AT_DELIMITER	非エスケープ引用符が検出されたときの処理戦略。 STOP_AT_DELIMITER：区切り文字で読み取りを停止 BACK_TO_DELIMITER：区切り文字に戻る STOP_AT_CLOSING_QUOTE：次の引用符で読み取りを停止 SKIP_VALUE：この列のデータをスキップ RAISE_ERROR：エラーを報告

USING ORC

ORCデータテーブルでサポートされている構成は次のとおりです：

OPTIONSでサポートされているキー	key に対応する value のデフォルト値	意味
圧縮またはorc.compress	snappy	圧縮アルゴリズムは、snappy/zlib/lzo/lz3/zstdの略称をサポートし、このパラメータはspark.sql.orc.compression.codecの影響を受けますが、テーブルパラメータが優先されます。
mergeSchema	false	スキーマをマージします。このパラメータはspark.sql.orc.mergeSchemaの

		影響を受け、テーブルパラメータが優先されます

HiveRead と HiveWriter（spark.sql.hive.convertMetastoreOrc=false を設定）を使用して読み書きする場合、OPTIONS は Orc ネイティブの設定もサポートしています。詳細は LanguageManual ORC を参照してください。

USING PARQUET

PARQUETデータテーブル関連のパラメータのほとんどはSpark confで設定可能であり、Spark confからの設定が推奨されます。optionsでも以下の設定がサポートされています：

OPTIONSでサポートされているキー	key に対応する value のデフォルト値	意味
圧縮または parquet.compression	snappy	圧縮アルゴリズムは、デフォルトでsnappyを使用し、パラメータspark.sql.parquet.compression.codecの影響を受けますが、テーブルパラメータが優先されます。
mergeSchema	false	スキーマをマージするかどうかは、パラメータspark.sql.parquet.mergeSchemaの影響を受け、テーブルパラメータが優先されます。
datetimeRebaseMode	EXCEPTION	parquetファイルを書き込む際の日付変換戦略。LEGACYモードでは日付をグレゴリオ暦に変換し、CORRECTEDでは日付をグレゴリオ暦に変換せず、EXCEPTIONでは日付が異なる形式の場合にエラーが発生します。パラメータspark.sql.parquet.datetimeRebaseModeInReadの影響を受けますが、テーブルパラメータが優先されます。
int96RebaseMode	EXCEPTION	parquetファイルを読み込む際の時間変換戦略。LEGACY

		モードでは時間をグレゴリオ暦に変換し、CORRECTEDでは時間を変換せず、EXCEPTIONでは異なる形式の時間の場合にエラーが発生します。パラメータspark.sql.parquet.int96RebaseModeInReadの影響を受けますが、テーブルパラメータが優先されます。

HiveRead と HiveWriter（spark.sql.hive.convertMetastoreParquet=false を設定）を使用して読み書きする場合、OPTIONS は Parquet ネイティブの設定もサポートしています。詳細は Hadoop integration を参照してください。

例

```
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');
```

よくあるご質問

CREATE_TABLE 時のキーワードでは、Spark の USING と Hive の STORED AS に違いがあり、テーブル作成後のファイル形式や読み取りが期待通りにならない可能性があります。ここで特別に説明します：

- USING DATA_SOURCE：Spark 構文。このキーワードは、テーブル作成時にどのデータソースを入力形式として使用するかを示し、テーブルの Location 下のファイル形式と読み取り方法に直接影響します。CSV、TXT、Iceberg、Parquet、Orc などの値を取ることができます。
- STORED AS FILE_FORMAT：Hive 構文。このキーワードは、HIVE 形式のテーブルを作成するために使用され、テーブルに保存されるデータファイルの形式を示します。TXT、Parquet、Orc などの値を取ることができます。この構文は使用しないことをお勧めします。Spark ネイティブの reader/writer がサポートされない可能性があります（例：CSV がサポートされない）。

ネイティブテーブル Iceberg 構文

> ⚠ 注意
> この構文はネイティブテーブルの作成のみをサポートします。

構文

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

パラメータ

`table_identifier`：3段式をサポート、catalog.db.name

スキーマとデータ型

```
col_type
  : primitive_type
  | nested_type

primitive_type
  : boolean
  | int/integer
  | long/bigint
  | float
  | double
  | decimal(p,s), p=最大桁数, s=最大小数点桁数, s<=p<=38
  | date
```

```
  | timestamp, timestamp with timezone, timeとwithout timezoneはサポートされ
ていません
  | string, Icebergのuuidタイプにも対応可能
  | binary、Iceberg fixed型にも対応可能

nested_type
  : struct
  | list
  | map
```

Partition Transforms

```
transform
  : identity、任意の型をサポート、DLCはこの変換をサポートしていません
  | bucket[N]、hash mod Nバケット分割、col_type: int、long、decimal、date、
timestamp、string、binaryをサポート
  | truncate[L]、Lで切り捨てバケット分割、col_type: int、long、decimal、string
をサポート
  | years、年、col_type: date、timestampをサポート
  | months、月、col_type: date、timestampをサポート
  | days/date、日付、col_type: date、timestampをサポート
  | hours/date_hour、時間、col_type: timestampをサポート
```

例

```
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

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：テーブル history を保持したまま、ターゲットテーブルからスナップショットテーブル snapshot を更新・置換します。

構文構造

```
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
```

例

```
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

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：データベース内のすべての基本テーブルとビューを一覧表示します。

構文

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

パラメータ

- `[IN database_name]`：テーブルを一覧表示するデータベース名を指定します。省略した場合、現在のコンテキストのデータベースが仮定されます。
- `['regular_expression']`：テーブルリストを指定された正規表現に一致するテーブルにフィルタリングします。任意の文字を表すワイルドカード*、または文字間を表すもののみ使用できます。

例

```
SHOW TABLES IN sampledb;
```

```
SHOW TABLES IN sampledb '*flights*';
```

SHOW CREATE TABLE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：存在するテーブル table_name を分析し、作成情報をクエリします。

構文構造

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

パラメータ

`TABLE [db_name.]table_name`

- `db_name`：データベース名。
- `table_name`：テーブル名。

例

```
SHOW CREATE TABLE tbl;
```

SHOW TBLPROPERTIES

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：命名テーブルのテーブル属性をリストします。

構文

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

パラメータ

`[('property_name')]`：含まれている場合、属性 name と value 値のみをリストします。

例

```
SHOW TBLPROPERTIES tb1;
```

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

DESCRIBE TABLE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：データテーブルの列情報とメタデータ情報を表示します。

標準構文

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

パラメータ

- `[EXTENDED | FORMATTED]`：指定されたテーブルのフォーマット形式。
- `table_name`：必要なテーブル名。
- `[PARTITION partition_spec]`：指定されたテーブルのパーティションリスト。

例

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

SHOW COLUMNS IN TABLE

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：データテーブルの列情報を表示します。

構文

```
SHOW COLUMNS IN table_name
```

パラメータ

`table_name`：データテーブル名。

例

```
SHOW COLUMNS IN clicks;
```

ALTER TABLE
ALTER TABLE ADD COLUMNS

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルの属性を変更します。

標準構文

```
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]
```

パラメータ

- `table_name`：必要なテーブル名。
- `partition_col1_name`：パーティション名。
- `partition_col1_value`：パーティション値。
- `col_name`：追加する列名。
- `data_type`：追加する列の型。

例

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

ALTER TABLE events ADD COLUMNS (eventowner string) CASCADE;

//ALTER TABLE PARTITION ADD COLUMNS構文はDLCネイティブテーブルのみをサポートしています
ALTER TABLE events PARTITION (year='2021') ADD COLUMNS (event string);
```

注意

テーブル作成時に `ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'` という形式で保存するテーブルを採用した場合、テーブル作成後に列を追加することはできません。JsonSerDe方式でテーブルを作成する際は、できるだけテーブルの構造を確認してください。列を追加する必要がある場合は、テーブルを削除して再作成することを検討してください。

ALTER TABLE ADD COLUMN AFTER/FIRST

最終更新日：：2025-12-25 11:51:48

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルに列を追加します。

標準構文

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

パラメータ

- `table_name`：変更が必要なテーブル名。
- `column_name1`：追加が必要な列。
- `column_type`：追加する列のタイプ。
- `col_comment`：追加する列のコメント。
- `column_name2`：追加する列をこの列の後ろに配置します。

例

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

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

ALTER TABLE DROP COLUMN

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルの特定のフィールドを削除します。

標準構文

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

パラメータ

- `table_name`：変更が必要なテーブル名。
- `column_name`：削除が必要な列名。

例

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

ALTER TABLE ADD PARTATION

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：外部テーブル。
- 用途：データテーブルに対して1つ以上のパーティション列を作成します。

構文構造

```
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, ...
```

パラメータ

- `table_name`：必要なテーブル名。
- `partition_column`：パーティション名。
- `partition_col_value`：パーティション値。

例

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

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

SHOW PARTITIONS

最終更新日：：2025-12-25 11:51:49

テーブル内のすべてのパーティションをリストします。

構文構造

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

パラメータ

- `TABLE [db_name.]table_name`：テーブル名。
- `[PARTITION(partition_spec)]`：パーティション列。複数のパーティション列条件を指定できます。

例

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

ALTER TABLE DROP PARTITION

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：外部テーブル。
- 用途：データテーブルの特定のパーティション列を削除します。

標準構文

```
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, ...
```

パラメータ

`table_name`：必要なテーブル名。
`partition_column`：パーティション名。
`partition_col_value`：パーティション値。

例

```
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

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルに単一のパーティションフィールドを追加します。

標準構文

```
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, …
```

パラメータ説明

- `table_name`：必要なテーブル名。
- `partition_column`：パーティション列。
- `alias`：パーティション列に追加する別名。

例

```
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

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルの特定のパーティションフィールドを削除します。

標準構文

```
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, …
```

パラメータ説明

- `table_name`：必要なテーブル名。
- `partition_column`：パーティション列。

例

```
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 ... RENAME COLUMN

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：フィールド名を変更します。

構文

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

パラメータ

- `table_identifier`：データテーブル名。
- `old_column_name`：変更が必要なフィールド名。
- `new_column_name`：変更後のフィールド名。

例

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

ALTER TABLE SET TBLPROPERTIES

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブル属性の更新/削除。

SET：属性構成の更新

構文

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

パラメータ

- `table_name`：必要なテーブル名。
- `property_name`：変更が必要なプロパティ名。
- `property_value`：変更が必要なプロパティ値。

例

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

UNSET：プロパティ設定を削除する

構文

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

パラメータ

- `table_name`：必要なテーブル名。
- `property_name`：変更が必要なプロパティ名。

例

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

ALTER TABLE SET LOCATION

最終更新日：：2025-12-25 11:51:49

説明

- サポート対象カーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルの保存パスを変更します。

標準構文

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

パラメータ

- `table_name`：データテーブルの名称。
- `PARTITION (partition_spec)`：パーティション列を指定します。
  - partition_col_name：パーティション列名。
  - partition_col_value：パーティション列の値。
- `'new location'`：tencent cos上の新しいテーブルまたはパーティションの場所。

例

```
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

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：SparkSQL。
- 適用テーブル範囲：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：テーブルデータ挿入時のソート方法を設定します。

構文

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

パラメータ

`table_identifier`：データテーブル名

例

```
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

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：パーティションテーブルのデータ配分戦略を変更する。

構文

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

例

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

ALTER TABLE ... SET IDENTIFIER FIELDS

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：identifier fields 属性を追加します。

構文

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

例

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

ALTER TABLE ... DROP IDENTIFIER FIELDS

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。
- 用途：identifier fields属性を削除します。

構文構造

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

例

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

MSCK REPAIR TABLE

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：データテーブルのパーティション情報を更新します。

標準構文

```
MSCK REPAIR TABLE table_identifier
```

パラメータ

`table_identifier`：テーブルの名前。

例

```
MSCK REPAIR TABLE t1
```

ANALYZE TABLES

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：データベーステーブルの統計をサポートします。

文法

```
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 ]
```

パラメータ

- `database_name`：統計情報を計算する必要があるテーブルが存在するデータベース。
- `table_identifier`：統計情報を計算する必要があるテーブル名。
- `partition_col_name`：統計情報を計算する必要があるパーティション列名。
- `partition_col_value`：統計情報を計算する必要があるパーティション列の値。

例

```
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

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブテーブル、外部テーブル。
- 用途：メタデータテーブルを削除します。

構文

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

パラメータ

- `IF EXISTS`：オプション、存在する場合の意味。
- `table_name`：テーブル名。

例

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

EXPLAIN

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- サポートテーブルタイプ：ネイティブテーブル、外部テーブル。
- 用途：sql実行の論理または物理プランを表示します。

構文

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
```

例

```
-- 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

最終更新日：：2025-12-25 11:51:49

ここでは、Sparkで Iceberg SQL拡張 を使用する場合にのみ、ストアドプロシージャが利用可能です。

構文

```
CALL expression
```

パラメータ

`expression`：関数式。

例

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

#位置引数で引数を渡す場合、それらがオプションであれば、末尾の引数のみ省略できます。
CALL catalog_name.system.procedure_name(arg_1, arg_2, ... arg_n)

#現在のスナップショットをdb.sample1に設定します
CALL catalog_name.system.set_current_snapshot('db.sample', 1)
```

さらに使用

Spark Procedures 。

CREATE VIEW AS

最終更新日：：2025-12-25 11:51:49

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビューの作成。

標準構文

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

パラメータ

- `[IF NOT EXISTS]`：存在しない場合は作成します。
- `view_name`：ビュー名。
- `[(column_name [COMMENT 'column_comment'][, ...])]`：列の名前。後ろに列のコメントを付けることができます。
- `[COMMENT 'view_comment']`：ビューのコメント。
- `select_statement`：クエリ文。

例

```
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

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：指定されたデータベース内のビューをリストします。データベース名を省略した場合、現在のデータベース内のビューがリストされます。

構文

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

パラメータ

- `[IN database_name]`：ビューをリストする対象のデータベース名を指定します。省略した場合、現在のコンテキストにおけるデータベースが仮定されます。
- `-`：ビューリストを指定された正規表現に一致するビューにフィルタリングします。任意の文字を表すワイルドカード*、または文字間の選択肢を表すもののみ使用できます。

例

```
SHOW VIEWS;

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

DESCRIBE VIEW

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビューの属性を表示します。

構文

```
DESCRIBE [view_name]
```

パラメータ

view_name：ビュー名。

例

```
DESCRIBE view1;
```

SHOW CREATE VIEW

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビュー作成の文を表示します。

構文

```
SHOW CREATE VIEW view_name
```

パラメータ

`view_name`：ビュー名。

例

```
SHOW CREATE VIEW orders_by_date
```

SHOW COLUMNS IN VIEW

最終更新日：：2025-12-25 11:51:50

基本説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビューの列情報を表示します。

標準構文

```
SHOW COLUMNS IN view_name;
```

パラメータ

`view_name`：ビュー名。

例

```
SHOW COLUMNS IN view_test
```

ALTER VIEW
ALTER VIEW RENAME TO

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビュー名の変更。

標準構文

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

パラメータ

- `old_view_identifier`：変更前のビュー名。
- `new_view_identifier`：変更後のビュー名。

例

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

ALTER VIEW SET TBLPROPERTIES

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビューの属性を変更します。

標準構文

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

パラメータ

- `view_identifier`：属性を変更する必要があるビューの名前。
- `property_key`：属性名。
- `property_val`：属性値。

例

```
alter view view1 set tblproperties('comment' = 'view1')

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

DROP VIEW

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：ビューの削除。

構文

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

パラメータ

- `IF EXISTS`：オプション、存在する場合の意味。
- `view_name`：ビュー名。

例

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

CREATE FUNCTION

最終更新日：：2025-12-25 11:51:50

説明

サポートカーネル：Presto、SparkSQL。
用途：クラス名で実装された関数を作成します。

構文構造

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

-関数のサフィックスが「_udtf」の場合、UDTF関数として認識されます
CREATE FUNCTION [db_name].function_udtf AS class_name
 [USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri'] ];
```

パラメータ

- `[db_name.]function_name`：関数名。関数作成時に名前空間を `db_name` 配下に指定します。
- `class_name`：関数の実装クラス。
- `USING JAR|FILE|ARCHIVE 'file_uri'`：関数リソースのパス。

例

```
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

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：関数作成の構文を表示します。

構文

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

パラメータ

- `function_name`：関数名。
- `regex_pattern`：関数名をフィルタリングするための正規表現。

例

```
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

最終更新日：：2025-12-25 11:51:50

説明

- サポートカーネル：Presto、SparkSQL。
- 用途：カスタム関数を削除します。

構文

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

パラメータ

`function_name`：削除する必要がある関数名。

例

```
DROP FUNCTION IF EXISTS `FUNC`

DROP FUNCTION `FUNC
```

DML構文
INSERT STATEMENT

最終更新日：：2025-12-25 12:00:06

テーブルに新しい行レコードを挿入します。列名リストが指定されている場合、それらはクエリによって生成される列名リストと完全に一致する必要があります。列名リストに含まれていないテーブル内の各列には、空の値が入力されます。列名リストが指定されていない場合、クエリによって生成される列は、挿入先のテーブルの列と完全に一致する必要があります。

構文

- Presto：

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

- Spark：

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

パラメータ

- `[ partition_spec ]`：パーティション列と値。例：dt='2021-06-01'。
- `[ ( column [, ... ] ) ]`：列のすべて。
- `[table_name] | table_identifier`：テーブル名。
- `[query]`：一般的なSelectクエリ文。

例

PrestoとSpark共通の挿入例：

```
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 サンプル:
パーティションを挿入するには、select クエリを使用します:

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

パーティションを挿入:

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

制限

Prestoはパーティションの挿入操作をサポートしていません。パーティションを挿入する必要がある場合は、sparkエンジンを使用して実行できます。

INSERT INTO

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：ソーステーブルで実行されるSELECTクエリの結果を新しい行としてターゲットテーブルに挿入することをサポートします。

構文

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

パラメータ

- `table_identifier`：テーブル名を指定します。三段式（例：`catalog.database.table`）がサポートされています。
- `partition_spec`：パーティション列と値。例：dt='2021-06-01'。
- `column_list`：列の一覧。
- `query`：汎用Selectクエリ文。
  1.1 `a SELECT statement`
  1.2 `a TABLE statement`

例

```
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

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：Presto、SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：行レベルのデータ挿入操作。

> **説明**
> PrestoはHiveデータソースのパーティションテーブルでのみinsert overwriteをサポートしており、非パーティションテーブルおよびIcebergデータソースのテーブルではこの使用法は現在サポートされていません。

構文

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

パラメータ

- `table_identifier`：テーブル名を指定します。カタログ.データベース.テーブルのような3段階形式をサポートしています
- `partition_spec`：パーティション列と値。例：dt='2021-06-01'。
- `column_list`：列のすべて。
- `query`：一般的なSelectクエリ文。
  1.1 a SELECT statement
  1.2 a TABLE statement

例

```
-- 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

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル、外部テーブル。
- 用途：行レベルのデータ更新操作で、INSERT OVERWRITE操作の代わりに使用できます。

構文

```
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 ]*
```

パラメータ

- `tablePrimary1`：テーブル名を指定します。カタログ.データベース.テーブルのような3段階の形式をサポートします。
- `alias`：エイリアス。
- `tablePrimary2`：テーブル名またはサブクエリを指定できます。
- `booleanExpression`：ブール式。

例

```
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

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル。
- 用途：データテーブルの指定行を更新します。

構文

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

パラメータ

- `tablePrimary`：テーブル名を指定します。カタログ.データベース.テーブルなどの3段階形式をサポートしています。
- `assign`：変更が必要な式。例：col1 = 'new_data'。
- `booleanExpression`：ブール式。

例

```
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

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：SparkSQL。
- 適用テーブル範囲：ネイティブIcebergテーブル。
- 用途：データテーブル内の指定行を削除します。

構文構造

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

パラメータ説明

`table_identifier`：テーブル名を指定します。カタログ.データベース.テーブルのような3段階形式をサポートしています

例

```
DELETE FROM lineitem WHERE shipmode = 'AIR';

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

DELETE FROM orders;
```

TABLE METADATA

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：ネイティブIcebergテーブル。
- 用途：Icebergテーブルのメタデータクエリを4段階（history、snapshots、files、manifests）でサポートします。

構文

```
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]
```

例

```
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構文
SELECT STATEMENT

最終更新日：：2025-12-25 12:00:06

`SELECT` ステートメント：ゼロ以上のテーブルからデータ行を取得します。

構文

```
[ 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 ] ]
```

パラメータ

[WITH with_query [, ...]]

WITHを使用してネストされたクエリをフラット化したり、サブクエリを簡素化したりできます。`with_query` の構文は次のとおりです：

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

- subquery_table_name は、WITH句のサブクエリ結果を定義するために使用される一時テーブルの一意の名前です。各subqueryは、FROM句で参照可能なテーブル名を持っている必要があります。
- column_name [, ...] はオプションの出力列名リストです。列名の数は、subqueryで定義された列数と等しいか、それ以下である必要があります。
- subquery は任意のクエリ文です。

[ALL | DISTINCT] select_expr

ALL と DISTINCT オプションは、重複行を返すかどうかを指定します。これらのオプションが指定されていない場合、デフォルトはALL（すべての一致する行を返す）です。DISTINCT は結果セットから重複行を削除すること

を指定します。

FROM from_item [, ...]

from_item はビュー、テーブル、サブクエリのいずれかであり、複数のテーブルを結合する場合、サポートされる結合タイプは次のとおりです:

- `[ INNER ] JOIN`
- `LEFT [ OUTER ] JOIN`
- `RIGHT [ OUTER ] JOIN`
- `FULL [ OUTER ] JOIN`
- `CROSS JOIN`
- `ON join_condition` 、 `join_condition` を使用する場合、複数のテーブルの結合キーに対して列名を指定できます； `join_column` を使用する場合、join_columnは両方のテーブルに存在する必要があります。

[WHERE condition]

指定した condition に基づいて結果をフィルタリングし、条件を満たす結果セットを返します。

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]]

GROUP BY 式は、指定した列名に基づいて出力をグループ化できます。

[HAVING condition]

集約関数とGROUP BY句と一緒に使用します。どのグループを選択するかを制御し、conditionを満たさないグループを除外します。このフィルタリングは、グループと集計の計算後に発生します。

[{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] union_query]

`UNION` 、 `INTERSECT` および `EXCEPT` は複数の結果を組み合わせます。 `UNION` は最初のクエリで生成された行と2番目のクエリで生成された行を組み合わせます。重複を排除するために、UNIONはハッシュテーブルを構築し、これによりメモリが消費されます。より良いパフォーマンスを得るためには、UNION ALLの使用が推奨されます。

- `INTERSECT` は、最初のクエリと2番目のクエリの結果の両方に存在する行のみを返します。
- `EXCEPT` は、最初のクエリ結果の行から、2番目のクエリで見つかった行を除いたものを返します。

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

1つ以上の出力式に基づいて結果セットを並べ替えます。句に複数の式が含まれている場合、結果セットは最初の `式` に基づいて並べ替えられます。次に、2番目の `式` が最初の式の一致する値を持つ行に適用され、以降も同様に続きます。

例

WITH Clause

WITH句は、クエリで使用される名前付きリレーションを定義します。これにより、ネストされたクエリをフラット化したり、サブクエリを簡素化したりできます。例えば、以下のクエリは同等です:

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

複数のサブクエリと組み合わせることもできます:

```
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

GROUPBY句は、SELECT文の出力を一致する値を持つ行グループに分割します。単純なGROUPBY句には、入力列で構成される任意の式を含めることができ、出力列を位置で選択する序数にすることもできます:

```
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

グループ化セットを使用すると、グループ化する列の複数のリストを指定できます。指定されたグループ化列のサブリストに含まれていない列は空に設定されます。

```
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

HAVING句は、集計関数とgroupby句と共に使用され、どのグループを選択するかを制御します。HAVING句は、指定された条件を満たさないグループを除外します。

```
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

IN 演算子は、WHERE 句で複数の値を指定することができます。

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

EXISTS

EXISTS演算子は、サブクエリにレコードがあるかどうかを判断するために使用されます。1つ以上のレコードが存在する場合はTrueを返し、それ以外の場合はFalseを返します。

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

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

USING

using キーワードを使用して簡素化します。

- クエリは等価条件の結合でなければなりません。
- 等価結合の列は同じ名前とデータ型を持っている必要があります。

```
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

クロス結合は、2つのリレーションのデカルト積（すべての組み合わせ）を返します。

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

LIMIT Clause

LIMIT句は結果セットの行数を制限します。

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

ORDER BY Clause

ORDER BY 句は、1つ以上の出力式に基づいて結果セットを並べ替えるために使用されます。

```
構文: ORDER BY 式 [ 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

EXCEPT句/演算子は、2つのSELECT文を結合し、2番目のSELECT文で返されなかった最初のSELECT文の行を返すために使用されます。これは、EXCEPTが2番目のSELECT文で利用できない行のみを返すことを意味します。
UNION操作を使用する場合と同様のルールが、EXCEPT演算子を使用する場合にも適用されます。

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

INTERSECT

SELECT ステートメントから2つ以上の結果セットの異なる行を返します。

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

UNION

2つ以上のSELECT文の結果セットを1つの結果セットに結合します。結果セット内の重複行を保持するには、`UNION ALL` 演算子を使用してください。

```
SELECT 13
UNION
SELECT 42
```

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

TABLESAMPLE

- `BERNOULLI`：各行をサンプルとして選択する確率はサンプルパーセンテージに等しい。Bernoulli法でテーブルをサンプリングする場合、テーブルのすべての物理ブロックをスキャンし、一部の行をスキップします（サンプリングパーセンテージと実行時に計算されるランダム値の比較に基づく）。結果に行が含まれる確率は他の行とは独立しています。これにより、サンプリングテーブルをディスクから読み取るのに必要な時間は短縮されません。サンプリング出力をさらに処理する場合、総クエリ時間に影響を与える可能性があります。
- `SYSTEM`：このサンプリング方法は、テーブルを論理データセグメントに分割し、その粒度でテーブルをサンプリングします。このサンプリング方法では、特定のデータセグメントからすべての行を選択するか、スキップします（サンプリングパーセンテージと実行時に計算されるランダム値の比較に基づく）。システムサンプリングで選択される行は、使用するコネクタによって異なります。たとえば、Hiveと一緒に使用する場合、HDFS上のデータのレイアウト方法に依存します。この方法では、独立したサンプリング確率は保証されません。

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

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

PIVOT Clause

特定の列に基づいて集計値を返します。

```
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 [ , ... ]
```

エスケープ

シングルクォートをエスケープするには、その前に別のシングルクォートを追加します。以下の例のように:

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

テーブルを作成する際にエスケープ文字またはエスケープシーケンスを指定します。例えば、以下の方法で作成します:

```
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 テーブル構文」DDL 構文

最終更新日：：2025-12-25 12:00:06

説明：
以下の構文説明はDLCネイティブテーブルの構文であり、DLCネイティブテーブルはデフォルトでIcebergテーブルです。Iceberg外部テーブルを使用する場合、DDL構文に細かい違いがありますので、ドキュメントIceberg外部テーブルとネイティブテーブルの構文の違いを参照してください。

CREATE TABLE

構文

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

パラメータ

table_identifier

`table_identifier` は三段式をサポートしています：`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), p=最大桁数, s=最大小数点桁数, s<=p<=38
| date
```

```
| timestamp, timestamp with timezone, timeとwithout timezoneはサポートされて
いません
| string, Icebergのuuidタイプにも対応可能
| binary, Icebergのfixedタイプにも対応可能

nested_type
: struct
| list
| map
```

Partition Transforms

```
transform
: identity, 任意のタイプをサポート, DLCはこの変換をサポートしていません
| bucket[N]、ハッシュmod Nバケット、col_typeをサポート: int、long、decimal、
date、timestamp、string、binary
| truncate[L], Lによるバケット分割の切り捨て、サポートされるcol_type:
int,long,decimal,string
| years, 年、サポートされるcol_type: date,timestamp
| months, 月、サポートされるcol_type: date,timestamp
| days/date, 日付、サポートされるcol_type: date,timestamp
hours/date_hour, 時間, col_type: timestampをサポート
```

例

```
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), ident
ity(bno), bucket(3,num),truncate(10,data));
```

CREATE TABLE AS SELECT

構文構造

```
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
```

例

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

REPLACE TABLE AS SELECT

テーブルの履歴 History を保持したまま、SELECT クエリの結果を使用してスナップショット Snapshot を生成し、テーブルを更新します。

構文構造

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

例

```
CREATE OR REPLACE TABLE dempts_replace
    COMMENT 'table create as replace'
    PARTITIONED BY (eno, dno)
```

```
AS SELECT * from dempts;
```

DROP TABLE

構文構造

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

ALTER TABLE

テーブル構文の変更

ALTER TABLE ... RENAME TO

テーブル名の変更

構文構造

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

ALTER TABLE ... SET / UNSET TBLPROPERTIES

テーブルプロパティの更新/削除

構文構造

```
-- SET プロパティ構成の更新
ALTER TABLE table_identifier
SET TBLPROPERTIES (property_name=property_value, ...)

-- UNSET プロパティ構成の削除
ALTER TABLE table_identifier
UNSET TBLPROPERTIES (property_name, ...)
```

例

```
-- SET プロパティ構成の更新
ALTER TABLE dempts SET TBLPROPERTIES ('read.split.target-
size'='268435456');

-- UNSET プロパティ構成の削除
```

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

ALTER TABLE ... WRITE ORDERED BY

テーブルデータ挿入時のソート方法を設定

構文構造

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

例

```
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

パーティションテーブルのデータ配分戦略を変更

構文

```
ALTER TABLE table_identifier
WRITE DISTRIBUTED BY PARTITION
[ LOCALLY ORDERED BY
```

```
{col_name [ASC|DESC] [NULLS FIRST|LAST]}[, ...]]
```

例

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

ALTER TABLE COLUMNS

フィールド構文の変更

ALTER TABLE ... ADD COLUMNS

複数フィールドの追加

構文

```
-複数フィールドの追加
ALTER TABLE table_identifier
ADD COLUMNS (col_name col_type [COMMENT col_comment], ...)

-- 単一フィールドの追加
ALTER TABLE table_identifier
ADD COLUMN col_name col_type [COMMENT col_comment]
[FIRST | AFTER target_col_name]
```

例

```
-複数フィールドの追加
ALTER TABLE dempts
ADD COLUMNS (
    new_column_1 string comment 'new_column_1 docs',
    new_column_2 int comment 'new_column_2 docs'
);

-- 単一フィールドの追加
ALTER TABLE dempts
ADD COLUMN new_column_3 string comment 'new_column docs';
```

ALTER TABLE ... RENAME COLUMN

フィールド名の変更

構文

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

ALTER TABLE ... ALTER COLUMN

フィールドのタイプ/備考情報の変更

構文

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

現在、Iceberg TYPEの変更は、フィールドタイプの安全な拡張のみをサポートしています:

- int/integer –> long/bigint
- float –> double
- decimal(P,S) → decimal(P2,S)、ただしP2 > P、つまり精度が向上

例

```
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

テーブルフィールドを削除

構文構造

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

ALTER TABLE PARTITIONS

ALTER TABLE ... ADD PARTITION FIELD

単一パーティションフィールドの追加

構文構造

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

transform はCREATE TABLEの説明を参照してください。

例

```
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

単一パーティションフィールドの置換

構文構造

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

ALTER TABLE ... DROP PARTITION FIELD

単一パーティションフィールドを削除

構文構造

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

transform はCREATE TABLEの説明を参照してください。

例

```
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

identifier fields 属性を追加する

構文構造

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

ALTER TABLE .. DROP IDENTIFIER FIELDS

identifier fields 属性を追加する

構文構造

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

DML構文

最終更新日：：2025-12-25 12:00:06

INSERT OVERWRITE | INTO

行レベルのデータ挿入操作

文法

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

例

```
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

行レベルデータ更新操作は、INSERT OVERWRITE操作を置き換えるために使用できます

文法

```
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

文法

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

UPDATE

Spark 3.1以降、UPDATE操作がサポートされています

文法

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

例

```
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構文

最終更新日：：2025-12-25 12:00:06

SELECT

構文

```
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

4段階のIcebergテーブルメタデータクエリをサポートします。これには、`history`、`snapshots`、`files`、`manifests`が含まれます。

構文

```
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]
```

例

```
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

Spark3.3以上でサポートされており、文字列とUnixタイムスタンプの2つの形式をサポートしています

例

```
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 (Spark 3.3でサポート)

文字列とスナップショットidをサポート

例

```
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

例

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

SNAPSHOT_ID_XXXX

例

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

Procedure

最終更新日：：2025-12-25 12:00:06

説明

- サポートカーネル：SparkSQL。
- 適用テーブルタイプ：外部Icebergテーブル、ネイティブIcebergテーブル。

基本構文

```
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);
```

スナップショット管理

rollback_to_snapshot

スナップショットを指定バージョンにロールバック。
入力パラメータ：テーブル名とバージョン番号。

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

rollback_to_timestamp

スナップショットを指定タイムスタンプにロールバック。
入力パラメータ：テーブル名とタイムスタンプ。

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

set_current_snapshot

スナップショットの現在のバージョンを設定。
入力パラメータ：テーブル名とバージョン番号。

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

cherrypick_snapshot

指定されたスナップショットバージョンから現在のスナップショットにチェリーピックします。

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

メタデータ管理

expire_snapshots

期限切れのスナップショットをクリーンアップし、小ファイルの数を削減します。

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

例：

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

remove_orphan_files

参照されなくなったメタデータファイルを削除します。

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

例：

```
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

参照されなくなったメタデータファイルを削除します。

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

例:

```
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

データファイルのマージと書き換え、つまり小さなデータファイルのマージです。

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

例:

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

rewrite_manifests

マニフェストファイルのマージと書き換え。

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

例:

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

ancestors_of

スナップショットの血縁情報を取得します。

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

例:

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

データテーブル移行管理

> **注意**
> 元テーブルはHiveテーブルまたはSparkテーブルでなければなりません。

snapshot

元のテーブルに基づいて軽量の一時テーブルを作成し、一時テーブルは直接元のテーブルのスナップショットを再利用します。

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

例：

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

migrate

テーブル属性を更新して置き換えます。

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

例：

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

add_files

hiveから直接データファイルをロードし、データファイルを指定したパーティションに指定できます。

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

例：

```
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'));
```

Iceberg外部テーブルとネイティブテーブルの構文の違い

最終更新日：：2025-12-25 12:00:06

データレイクコンピューティング DLC で使用される Iceberg の構文バージョンは0.13.1です。詳細な構文説明については、Iceberg 公式ドキュメント を参照してください。
Iceberg外部テーブルを使用する場合、SQL構文はIcebergネイティブテーブルと以下の違いがあります。

CREATE TABLE

ネイティブテーブル

構文

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

例

```
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));
```

外部テーブル

構文

```
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, ... ) ]
```

例

```
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

ネイティブテーブル

構文

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

例

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

外部テーブル

構文

```
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
```

例

```
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

ネイティブテーブル

構文

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

例

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

外部テーブル

構文

```
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
```

例

```
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

> ⚠ 注意
> 移行元テーブルはHiveテーブルまたはSparkテーブルである必要があります。

ネイティブテーブル

サポートしていません。

外部テーブル

- **スナップショット**
  元のテーブルに基づいて軽量な一時テーブルを作成し、一時テーブルは直接元のテーブルのスナップショットを再利用します。
  構文

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

  例

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

- **call**
  テーブル属性を更新して置き換えます。
  構文

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

  例

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

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

- **add_files**
  hiveから直接データファイルをロードし、データファイルを指定したパーティションに割り当てることができます。
  構文

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

  例

```
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'));
```

マテリアライズドビュー構文

最終更新日：：2025-12-25 12:00:06

> **説明：**
> 以下はマテリアライズドビューでサポートされている構文で、PrestoおよびSparkSQLエンジンのみで利用可能です。詳細な使用ガイドはマテリアライズドビュードキュメントを参照してください。

CREATE MATERIALIZED

マテリアライズドビューを作成：

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] mv_identifier
    [ENABLE | DISABLE REWRITE]
    [COMMENT mv_comment]
    [WITH META LINK]
    [TBLPROPERTIES (key1=value1, key2=value2,...)]
    [AS] select_query
```

DROP MATERIALIZED

マテリアライズドビューを削除：

```
DROP MATERIALIZED VIEW [IF EXISTS] mv_identifier
```

DESCRIBE MATERIALIZED

マテリアライズドビューを記述：

```
[DESCRIBE | DESC] MATERIALIZED VIEW mv_identifier
```

REFRESH MATERIALIZED

マテリアライズドビューをリフレッシュ：

```
REFRESH MATERIALIZED VIEW mv_identifier
```

SHOW MATERIALIZED VIEW JOBS

マテリアライズドビューの実行タスクリストを表示し、マテリアライズドビューのリフレッシュ履歴と状態を確認できます。

```
SHOW MATERIALIZED VIEW JOBS IN mv_identifier
```

SQL暗黙の型変換

最終更新日：：2025-12-25 12:00:06

SQLの作成を容易にするため、Hive/Spark構文からDLC統一SQLへの迅速な切り替えを可能にするために、SQLの暗黙的変換機能を提供しています。変換リストは以下の通りです：
I：暗黙的キャスト、E: 明示的キャスト、--: 許可されていません

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

関数
統一関数
統一関数の概要

最終更新日：：2025-12-25 12:00:07

データレイクコンピューティング DLC は、統一関数を介して異なるカーネルで使用することをサポートし、Spark や Presto とも互換性があります。具体的な関数とサポートされているカーネルについては、以下の表を参照してください。
データレイクコンピューティング DLC は、Presto の組み込み関数もサポートしています。サポートリストと有効化方法については、Presto 組み込み関数 を参照してください。

関数	Spark	Presto
ABS	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：入力パラメータと一致	出参：double
ACOS	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
ACOSH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
ADD_MONTHS	入参：date|timestamp|string, int	入参：date|timestamp|string, int
	出参：date	出参：date
AES_DECRYPT	–	入参：(binary, string|binary)
		出参：binary
AES_ENCRYPT	–	入参：(string|binary, string|binary)
		出参：binary
AND	入参：boolean AND boolean	boolean AND boolean
	出参：boolean	出参：boolean
ANY	入参：boolean	入参：boolean

	出参：boolean	出参：boolean
ANY_MATCH	入参：(array<T>, lambda function)	–
	出参：boolean	
APPROX_COUNT_DISTINCT	入参：(bigint|double|decimal|date|timestamp)	–
	double|array<double>[, bigint])	
	出参：2番目の入力パラメータと同じ	
APPROX_PERCENTILE	入参：(bigint|double|decimal|, array<double>|double[, int])	入参：(bigint|double|decimal, double[, int])
	出参：double|array	出参：double|array
ARG_MAX	入参：　(col1, col2 | expr(col2))	–
	出参：col2またはexpr(col2)のタイプ	
ARG_MIN	入参：　(col1, col2 | expr(col2))	
	出参：col2またはexpr(col2)のタイプ	
ARRAY	入参：(T, ...)	入参：(T, ...)
	出参：array<T>	出参：array<T>
ARRAYS_OVERLAP	入参：(array<T>, array<T>)	入参：(array<T>, array<T>)
	出参：boolean	出参：boolean
ARRAYS_ZIP	入参：(array<T>, array<U>, ...)	–
	出参：array<struct<T, U, ...>>	
ARRAY_CONTAINS	入参：(array<T>, T)	入参：(array<T>, T)
	出参：boolean	出参：boolean
ARRAY_DISTINCT	入参：array<T>	入参：array<T>
	出参：array<T>	出参：array<T>
ARRAY_EXCEPT	入参：(array<T>, array<T>)	入参：(array<T>, array<T>)
	出参：array<T>	出参：array<T>

ARRAY_INTERSECT	入参：(array<T>, array<T>)	入参：(array<T>, array<T>)
	出参：array<T>	出参：array<T>
ARRAY_JOIN	入参：(array<T>, string[, string])	入参：(array<T>, string[, string])
	出参：string	出参：string
ARRAY_MAX	入参：array<bigint|double|decimal>	入参：array<bigint|double|decimal>
	出参：bigint|double|decimal	出参：bigint|double|decimal
ARRAY_MIN	入参：array<bigint|double|decimal>	入参：array<bigint|double|decimal>
	出参：bigint|double|decimal	出参：bigint|double|decimal
ARRAY_POSITION	入参：(array<T>, bigint)	入参：(array<T>, bigint)
	出参：bigint	出参：bigint
ARRAY_REMOVE	入参：(array<T>, T)	入参：(array<T>, T)
	出参：array<T>	出参：array<T>
ARRAY_REPEAT	入参：(array<T>, bigint)	入参：(array<T>, bigint)
	出参：array<T>	出参：array<T>
ARRAY_SORT	入参：(array<T>, function(T, T)–>integer)	入参：(array<T>)
	出参：array<T>	出参：array<T>
ARRAY_UNION	入参：(array<T>, array<T>)	入参：(array<T>, array<T>)
	出参：array<T>	出参：array<T>
ASCII	入参：string	入参：string
	出参：int	出参：int
ASIN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
ASINH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double

ASSERT_TRUE	入参：boolean	入参：boolean
	出参：null|例外をスロー	出参：null|例外をスロー
ATAN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
ATAN2	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：double	出参：double
ATANH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
AVG	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
BASE64	入参：string|binary	入参：binary
	出参：string	出参：string
BIGINT	強制型変換をbigintに	強制型変換をbigintに
BIN	入参：bigint	入参：bigint
	出参：string	出参：string
BINARY	binary への強制型変換	binary への強制型変換
BIT_AND	入参：int	入参：int
	出参：int	出参：int
BIT_COUNT	入参：int|boolean	入参：int|boolean
	出参：int	出参：int
BIT_GET	入参：(int, int)	入参：(int, int)
	出参：int	出参：int
BIT_LENGTH	入参：string	入参：string
	出参：int	出参：int

BIT_OR	入参：int	入参：int
	出参：int	出参：int
BIT_XOR	入参：int	入参：int
	出参：int	出参：int
BOOLEAN	boolean への強制型変換	boolean への強制型変換
BOOL_AND	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
BOOL_OR	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
BROUND	入参：(bigint|double|decimal, int)	入参：(bigint|double|decimal, int)
	出参：int	出参：int
BTRIM	入参：(string[, string])	入参：(string[, string])
	出参：string	出参：string
CARDINALITY	入参：(array|map)	入参：(array|map)
	出参：int	出参：int
CAST	入参：(<expr> AS T)	入参：(<expr> AS T)
	出参：T	出参：T
CBRT	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
CEIL	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：bigint	出参：bigint
CEILING	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：bigint	出参：bigint
CHAR	入参：int	入参：int
	出参：string	出参：string

CHARACTER_LENGTH	入参：string|binary	入参：string
	出参：int	出参：int
CHAR_LENGTH	入参：string|binary	入参：string
	出参：int	出参：int
CHR	入参：int|double	入参：int|double
	出参：string	出参：string
CLUSTER_SAMPLE	入参：　(bigint [, bigint])	–
	出参：boolean	
COALESCE	入参：(T, T, ...)	入参：(T, T, ...)
	出参：T	出参：T
COLLECT_LIST	入参：T	入参：T
	出参：array<T>	出参：array<T>
COLLECT_SET	入参：T	入参：T
	出参：array<T>	出参：array<T>
CONCAT	入参：string|array	入参：(string|binary, string|binary, ...)
	出参：入力パラメータと一致	出参：string
CONCAT_WS	入参：(string, [string | array<string>]+)	入参：(string, [string | array<string>]+)
	出参：string	出参：string
CONTEXT_NGRAMS	–	入参：(array<array<string>>, array<string>, int, int)
		出参：array<struct<string, double>>
CONV	入参：(bigint|double|decimal|string, int, int)	入参：(bigint|double|decimal|string, int, int)
	出参：文字列	出参：文字列

CORR	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
	出参：double	
COS	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
COSH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
COT	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
COUNT	入参：T	入参：T
	出参：bigint	出参：bigint
COUNT_IF	入参：boolean	–
	出参：int	
COUNT_MIN_SKETCH	入参：(int|binary|string, double, double, int)	–
	出参：binary	
COVAR_POP	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
COVAR_SAMP	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
CRC32	入参：string|binary	入参：string|binary
	出参：bigint	出参：bigint
CUME_DIST	入参：なし	入参：なし
	出参：double	出参：double
CURRENT_CATALOG	入参：なし	–

	出参：文字列	
CURRENT_DATABASE	入参：なし	–
	出参：文字列	
CURRENT_DATE	入参：なし	入参：なし
	出参：date	出参：date
CURRENT_TIMESTAMP	入参：なし	入参：なし
	出参：timestamp	出参：timestamp
CURRENT_TIMEZONE	入参：なし	入参：なし
	出参：文字列	出参：文字列
CURRENT_USER	入参：なし	入参：なし
	出参：文字列	出参：文字列
DATE	入参：string	入参：string
	出参：date	出参：date
DATEDIFF	入参：(string|timestamp|date, string|timestamp|date)	入参：(string|timestamp|date, string|timestamp|date)
	出参：int	出参：int
DATE_ADD	入参：(string|date|timestamp, int)	入参：(string|date|timestamp, int)
	出参：date	出参：date
DATEADD	入参：(date/timestamp, int, string)	–
	出参：timestamp	
DATE_FORMAT	入参：(string|date|timestamp, string)	入参：(string|date|timestamp, string)
	出参：文字列	出参：文字列
DATE_FROM_UNIX_DATE	入参：int	入参：int
	出参：date	出参：date

DATE_PART	入参：(string, string|date|timestamp)	入参：(string, string|date|timestamp)
	出参：bigint|double|decimal	出参：bigint|double|decimal
DATE_SUB	入参：(string|date|timestamp)	入参：(string|date|timestamp)
	出参：date	出参：date
DATE_TRUNC	入参：(string, string)	入参：(string, string)
	出参：timestamp	出参：timestamp
DAY	入参：string|timestamp|date	入参：string|timestamp|date
	出参：int	出参：int
DAYOFMONTH	入参：string|timestamp|date	入参：string|timestamp|date
	出参：int	出参：int
DAYOFWEEK	入参：string|timestamp|date	string|timestamp|date
	出参：int	出参：int
DAYOFYEAR	入参：string|timestamp|date	string|timestamp|date
	出参：int	出参：int
DECIMAL	cast as decimal	cast as decimal
DECODE	入参：(binary, string) | (int, int, string[, int, string]...[, default])	入参：(binary, string)
	出参：int	出参：文字列
DEGREES	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
DENSE_RANK	入参：なし	入参：なし
	出参：int	出参：int
DIV	入参：bigint|double|decimal	–
	出参：int	
DOUBLE	cast as double	cast as double

E	入参：なし	入参：なし
	出参：double	出参：double
ELEMENT_AT	入参：(array|map, int)	入参：(array, int)
	出参：array要素|map valueと一致	出参：array要素と一致
ELT	入参：(int, T, U,...)	入参：(int, string, string, ...)
	出参：入力に依存	出参：文字列
ENCODE	入参：(string|binary, string)	入参：(string, string)
	出参：binary	出参：binary
EVERY	入参：T	入参：T
	出参：boolean	出参：boolean
EXP	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
EXPLODE	入参：array<T>	–
	出参：T	
EXPLODE_OUTER	入参：array<T>	–
	出参：T	
EXPM1	入参：bigint|double|decimal	–
	出参：double	
FACTORIAL	入参：int	入参：int
	出参：bigint	出参：bigint
FIELD	–	入参：(T, T, T, ...)
		出参：int
FIND_IN_SET	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
FIRST_VALUE	入参：(T[, boolean])	–

	出参：T	
FLATTEN	入参：array<array<T>>	入参：array<array<T>>
	出参：array<T>	出参：array<T>
FLOAT	cast as float	cast as float
FLOOR	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：bigint	出参：bigint
FORMAT_NUMBER	入参：(bigint|double|decimal, int|string)	入参：(bigint|double|decimal, int)
	出参：文字列	出参：文字列
FORMAT_STRING	入参：(string, T, ...)	入参：(string, T, ...)
	出参：文字列	出参：文字列
FROM_CSV	入参：(string, string[, map])	–
	出参：struct	
FROM_JSON	入参：(string, string[, map])	–
	出参：struct	
FROM_UNIXTIME	入参：(bigint[, string])	入参：(bigint[, string])
	出参：文字列	出参：文字列
FROM_UTC_TIMESTAMP	入参：(string, string)	入参：(bigint|double|decimal|timestamp|date|string, string)
	出参：timestamp	出参：timestamp
GETBIT	入参：(int, int)	入参：(int, int)
	出参：int	出参：int
GET_IDCARD_AGE	入参：string	–
	出参：int	
GET_IDCARD_BIRTHDAY	入参：string	–
	出参：date	

GET_IDCARD_SEX	入参：string	–
	出参：文字列	
GREATEST	入参：(bigint|double|decimal, bigint|double|decimal, ...)	入参：(bigint|double|decimal, bigint|double|decimal, ...)
	出参：入力と一致	出参：入力と一致
GROUPING	入参：T	T
	出参：int	出参：int
GROUPING_ID	入参：[T[, T...]]	–
	出参：int	
HASH	入参：(T, T,...)	入参：(T, T,...)
	出参：int	出参：int
HEX	入参：int|string|binary	入参：int|string|binary
	出参：文字列	出参：文字列
HISTOGRAM_NUMERIC	–	入参：bigint|double|decimal
		出参：array<struct<'x', 'y'>>
HOUR	入参：string|timestamp	入参：string|timestamp
	出参：int	出参：int
HYPOT	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
IF	入参：(boolean, T, T)	入参：(boolean, T, T)
	出参：T	出参：T
IFNULL	入参：(T, U)	–
	出参：T|U	
IN	入参：(T, T,...)	入参：(T, T,...)
	出参：boolean	出参：boolean

INITCAP	入参：string	入参：string
	出参：文字列	出参：文字列
INLINE	入参：array	–
	出参：table	
INLINE_OUTER	入参：array	–
	出参：table	
INPUT_FILE_BLOCK_LENGTH	入参：なし	–
	出参：int	
INPUT_FILE_BLOCK_START	入参：なし	–
	出参：int	
INPUT_FILE_NAME	入参：なし	–
	出参：文字列	
INSTR	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
INT	int への強制型変換	int への強制型変換
ISNAN	入参：T	入参：T
	出参：boolean	出参：boolean
ISNOTNULL	入参：T	入参：T
	出参：boolean	出参：boolean
ISNULL	入参：T	入参：T
	出参：boolean	出参：boolean
JSON_ARRAY_LENGTH	入参：string	入参：string
	出参：int	出参：int
JSON_OBJECT_KEYS	入参：string	入参：string
	出参：array	出参：array

JSON_TUPLE	入参：(string, ..., string) 出参：string...string	–
KEYVALUE	入参：(string [, string, string], string) 出参：文字列	
KURTOSIS	入参：bigint|double|decimal 出参：double	–
LAG	入参：(T[, int[, string]]) 出参：T	入参：(T[, int[, string]]) 出参：T
LAST_DAY	入参：string|date|timestamp 出参：文字列	入参：string|date|timestamp 出参：文字列
LAST_VALUE	入参：(T[, boolean]) 出参：T	–
LCASE	入参：string 出参：文字列	入参：string 出参：文字列
LEAD	入参：(T[, int[, string]]) 出参：T	入参：(T[, int[, string]]) 出参：T
LEAST	入参：(bigint|double|decimal, bigint|double|decimal, ...) 出参：入力と一致	(bigint|double|decimal, bigint|double|decimal, ...) 出参：入力と一致
LEFT	入参：(string, int) 出参：文字列	入参：(string, int) 出参：文字列
LENGTH	入参：string 出参：int	入参：string 出参：int
LEVENSHTEIN	入参：(string, string) 出参：int	入参：(string, string) 出参：int

LIKE	入参：(string, string) | string LIKE string ESCAPE string	入参：(string, string)
	出参：boolean	出参：boolean
LN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
LOCATE	入参：(string, string[, int])	入参：(string, string[, int])
	出参：int	出参：int
LOG	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：double	出参：double
LOG10	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
LOG1P	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
LOG2	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
LOWER	入参：string	入参：string
	出参：文字列	出参：文字列
LPAD	入参：(string, int[, string])	入参：(string, int, string)
	出参：文字列	出参：文字列
LTRIM	入参：string	入参：string
	出参：文字列	出参：文字列
MAKE_DATE	入参：(int, int, int)	入参：(int, int, int)
	出参：date	出参：date
MAKE_TIMESTAMP	入参：(int, int, int, int, int, int[, string])	入参：(int, int, int, int, int, int[, string])

	出参：timestamp	出参：timestamp
MAP	入参：(K1, V1, K2, V2,...)	入参：(array<K>, array<V>)
	出参：map	出参：map
MAP_CONCAT	入参：(map, ...)	入参：(map, ...)
	出参：map	出参：map
MAP_ENTRIES	入参：map	入参：map
	出参：array	出参：array
MAP_FROM_ARRAYS	入参：(array, array)	入参：(array, array)
	出参：map	出参：map
MAP_FROM_ENTRIES	入参：array	入参：array
	出参：map	出参：map
MAP_KEYS	入参：map	入参：map
	出参：array	出参：array
MAP_UINON_SUM	入参：map	–
	出参：map	
MAP_VALUES	入参：map	入参：map
	出参：array	出参：array
MAX	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：入力パラメータと一致	出参：double
MAX_BY	入参：T, bigint|double|decimal	入参：T, bigint|double|decimal
	出参：T	出参：T
MAX_PT	入参：const string	–
	出参：string	
MD5	入参：string|binary	入参：string|binary
	出参：文字列	出参：string

MEAN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
MIN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：入力パラメータと一致	出参：double
MINUTE	入参：(文字列|タイムスタンプ)	入参：(文字列|タイムスタンプ)
	出参：int	出参：int
MIN_BY	入参：(T, bigint|double|decimal)	入参：(T, bigint|double|decimal)
	出参：T	出参：T
MOD	入参：(bigint|double|decimal, bigint|double|decimal) | bigint|double|decimal MOD bigint|double|decimal	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：int|double	出参：int|double
MONOTONICALLY_INCREASING_ID	入参：なし	–
	出参：bigint	
MONTH	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int
MONTHS_BETWEEN	入参：(date|timestamp|string, date|timestamp|string[, boolean])	入参：(date|timestamp|string, date|timestamp|string)
	出参：double	出参：double
NAMED_STRUCT	入参：(K1, V1, K2, V2, ...)	–
	出参：struct	
NANVL	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：double	出参：double
NEGATIVE	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：bigint|double|decimal	出参：int|double

NEXT_DAY	入参：(string|date|timestamp, string)	入参：(string|date|timestamp, string)
	出参：date	出参：文字列
NGRAMS	-	入参：(array<array<string>>, int, int, int)
		出参：array<struct<string, double>>
NOW	入参：なし	入参：なし
	出参：timestamp	出参：timestamp
NTH_VALUE	入参：(T[, int])	入参：(T[, int])
	出参：T	出参：T
NTILE	入参：int	入参：int
	出参：int	出参：int
NULLIF	入参：(T, T)	入参：(T, T)
	出参：T	出参：T
NVL	入参：(T, T)	入参：(T, T)
	出参：T	出参：T
NVL2	入参：(T, T, T)	入参：(T, T, T)
	出参：T	出参：T
OCTET_LENGTH	入参：string	入参：string
	出参：int	出参：int
OR	入参：expr1 OR expr2	入参：expr1 OR expr2
	出参：boolean	出参：boolean
OVERLAY	入参：(string PLACING string FROM int[FOR int])	-
	出参：int	
PARSE_URL	入参：(string, string[, string])	入参：(string, string[, string])

	出参：string	出参：string
PERCENTILE	入参：(bigint|double|decimal, bigint|double|decimal|array, int)	入参：(bigint|double|decimal, bigint|double|decimal|array, int)
	出参：bigint|double|decimal|array	出参：double|array
PERCENTILE_APPROX	入参：(bigint|double|decimal, bigint|double|decimal|array[, int])	入参：(bigint|double|decimal, bigint|double|decimal[, int])
	出参：double|array	出参：double|array
PERCENT_RANK	入参：なし	入参：なし
	出参：double	出参：double
PI	入参：なし	入参：なし
	出参：double	出参：double
PMOD	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：入力パラメータと一致	出参：入力パラメータと一致
POSITION	入参：(string, string[, int]) | (string IN string)	入参：(string, string[, int]) | (string IN string)
	出参：int	出参：int
POSITIVE	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：入力パラメータと一致	出参：入力パラメータと一致
POSEXPLODE	入参：array	–
	出参：table	
POSEXPLODE_OUTER	入参：array	–
	出参：table	
POW	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：double	出参：double
POWER	入参：(bigint|double|decimal, bigint|double|decimal)	入参：(bigint|double|decimal, bigint|double|decimal)

	出参：double	出参：double
PRINTF	入参：(string, T, ...)	入参：(string, T, ...)
	出参：string	出参：string
QUARTER	入参：string|date|timestamp	入参：string|date|timestamp
	出参：int	出参：int
RADIANS	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
RAISE_ERROR	入参：string	入参：string
	出参：例外をスロー	出参：例外をスロー
RAND	入参：[int]	入参：[int]
	出参：double	出参：double
RANDN	入参：[int]	入参：[long]
	出参：double	出参：double
RANDOM	入参：[int]	入参：[int]
	出参：double	出参：double
RANK	入参：なし	入参：なし
	出参：int	出参：int
REGEXP	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
REGEXP_COUNT	入参：(string, string)	_
	出参：int	
REGEXP_EXTRACT	入参：(string, string[, int])	入参：(string, string, int)
	出参：int	出参：int
REGEXP_EXTRACT_ALL	入参：(string, string[, int])	入参：(string, string[, int])
	出参：array	出参：array

REGEXP_LIKE	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
REGEXP_REPLACE	入参：(string, string, string[, int])	入参：(string, string, string)
	出参：string	出参：string
REGEXP_SUBSTR	入参：(string, string)	-
	出参：string	
REPEAT	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
REPLACE	入参：(string, string[, string])	入参：(string, string, string)
	出参：string	出参：string
REVERSE	入参：string | array	入参：string
	出参：string	出参：string
RIGHT	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
RINT	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
RLIKE	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
ROUND	入参：(double, int)	入参：(double, int)
	出参：double	出参：double
ROW_NUMBER	入参：なし	入参：なし
	出参：int	出参：int
RPAD	入参：(string, int[, string])	入参：(string, int, string)
	出参：string	出参：string
RTRIM	入参：string	入参：string

	出参：string	出参：string
SAMPLE	入参：(long [, long [, cols...]])	–
	出参：boolean	
SCHEMA_OF_CSV	入参：(string[, map])	
	出参：struct	
SCHEMA_OF_JSON	入参：(string[, map])	–
	出参：int	
SECOND	入参：string|timestamp	入参：string|timestamp
	出参：int	出参：int
SENTENCES	(string[, string, string])	入参：(string[, string, string])
	出参：array<array<string>>	出参：array<array<string>>
SEQUENCE	入参：(int|bigint|date|timestamp, int|bigint|date|timestamp, int)	入参：(int|bigint|date|timestamp, int|bigint|date|timestamp, int)
	出参：入力パラメータと一致	出参：入力パラメータと一致
SHA	入参：string|binary	入参：string|binary
	出参：string	出参：string
SHA1	入参：string|binary	入参：string|binary
	出参：string	出参：string
SHA2	入参：(string, int)	入参：(string, int)
	出参：string	出参：string
SHIFTLEFT	入参：(int|bigint, int)	引数：(int|bigint, int)
	出参：int|bigint	出参：int|bigint
SHIFTRIGHT	入参：(int|bigint, int)	入参：(int|bigint, int)
	出参：int|bigint	出参：int|bigint
SHIFTRIGHTUNSIGNED	入参：(int|bigint, int)	入参：(int|bigint, int)

	出参：int|bigint	出参：int|bigint
SHUFFLE	入参：array	入参：array
	出参：array	出参：array
SIGN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
SIGNUM	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
SIN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
SINH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
SIZE	入参：array | map	入参：array | map
	出参：int	出参：int
SKEWNESS	入参：bigint|double|decimal	–
	出参：double	
SLICE	入参：(array, int, int)	入参：(array, int, int)
	出参：array	出参：array
SMALLINT	cast as smallint	cast as smallint
SOME	入参：boolean	入参：boolean
	出参：boolean	出参：boolean
SORT_ARRAY	入参：(array[, boolean])	入参：array
	出参：array	出参：array
SOUNDEX	入参：string	入参：string
	出参：string	出参：string
SPACE	入参：文字列	入参：文字列

	出参：文字列	出参：文字列
SPARK_PARTITION_ID	入参：なし 出参：int	–
SPLIT	入参：(string, string, int) 出参：文字列	入参：(string, string) 出参：文字列
SQRT	入参：bigint|double|decimal 出参：double	入参：bigint|double|decimal 出参：double
STACK	入参：(T, U, ...) 出参：table	–
STD	入参：bigint|double|decimal 出参：double	入参：bigint|double|decimal 出参：double
STDDEV	入参：bigint|double|decimal 出参：double	入参：bigint|double|decimal 出参：double
STDDEV_POP	入参：bigint|double|decimal 出参：double	入参：bigint|double|decimal 出参：double
STDDEV_SAMP	入参：bigint|double|decimal 出参：double	入参：bigint|double|decimal 出参：double
STRING	cast as string	cast as string
STRUCT	入参：(T, U, ...) 出参：struct	入参：(T, U, ...) 出参：struct
STR_TO_MAP	入参：(string[, string[, string]]) 出参：map	入参：(string[, string, string]) 出参：map
SUBSTR	入参：(string, int[, int]) | (string FROM int [FOR int]) 出参：文字列	入参：(string|binary, int[, int]) | (string|binary FROM int [FOR int]) 出参：string

SUBSTRING	入参：(string, int[, int]) | (string FROM int [FOR int])	入参：(string|binary, int[, int]) | (string|binary FROM int [FOR int])
	出参：文字列	出参：string
SUBSTRING_INDEX	入参：(string, string, int)	入参：(string, string, int)
	出参：文字列	出参：文字列
SUM	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：入力と一致	出参：double
TAN	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
TANH	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
TIMESTAMP	入参：文字列	入参：文字列
	出参：timestamp	出参：timestamp
TIMESTAMP_ADD	入参：(date/timestamp, int, string)	
	出参：timestamp	
TIMESTAMP_MICROS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMESTAMP_MILLIS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMESTAMP_SECONDS	入参：bigint	入参：bigint
	出参：timestamp	出参：timestamp
TIMEZONE_HOUR	入参：文字列	–
	出参：int	
TIMEZONE_MINUTE	入参：文字列	–
	出参：int	

TINYINT	cast as tinyint	cast as tinyint
TO_CHAR	入参：(boolean|int|long|double|decimal [, string])	
	出参：文字列	
TO_CSV	入参：(struct[, map])	–
	出参：文字列	
TO_DATE	入参：(string[, string])	入参：文字列
	出参：date	出参：date
TO_JSON	入参：(string[, map])	–
	出参：文字列	
TO_TIMESTAMP	入参：(string[, string])	入参：(string)
	出参：timestamp	出参：timestamp
TO_UNIX_TIMESTAMP	入参：(string[, string])	入参：(string[, string])
	出参：timestamp	出参：timestamp
TO_UTC_TIMESTAMP	入参：(string[, string])	入参：(bigint|double|decimal|date|timestamp|string, string)
	出参：timestamp	出参：timestamp
TRANS_ARRAY	入参：(int, string, cols...)	–
	出参：colsと同じタイプ	
TRANS_COLS	入参：(int, cols...)	
	出参：(int, cols)	
TRANSLATE	入参：(string, string, string)	–
	出参：文字列	
TRIM	入参：文字列	入参：文字列
	出参：文字列	出参：string

TRUNC	入参：(string|date|timestamp, string)	入参：(string|date|timestamp, string)
	出参：文字列	出参：string
TRY_ADD	入参：(bigint|double|decimal|date|timestamp|, bigint|double|decimal|date|timestamp)	入参：(bigint|double|decimal, bigint|double|decimal)
	出参：入力パラメータと一致	出参：入力パラメータと一致
TRY_DIVIDE	入参：(bigint|double|decimal, bigint|double|decimal)	–
	出参：double	
TYPEOF	入参：T	入参：T
	出参：文字列	出参：文字列
UCASE	入参：文字列	入参：文字列
	出参：文字列	出参：文字列
UNBASE64	入参：文字列	入参：文字列
	出参：文字列	出参：文字列
UNHEX	入参：文字列	入参：文字列
	出参：binary	出参：binary
UNIX_DATE	入参：string|timestamp|date	入参：string|timestamp|date
	出参：int	出参：int
UNIX_MICROS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint
UNIX_MILLIS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint
UNIX_SECONDS	入参：timestamp	入参：timestamp
	出参：bigint	出参：bigint

UNIX_TIMESTAMP	入参：[date|timestamp|string[, string]]	入参：(date|timestamp|string[, string])
	出参：bigint	出参：bigint
UPPER	入参：文字列	入参：文字列
	出参：文字列	出参：文字列
URL_DECODE	入参：文字列	–
	出参：文字列	
UUID	入参：なし	入参：なし
	出参：文字列	出参：文字列
VARIANCE	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
VAR_POP	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
VAR_SAMP	入参：bigint|double|decimal	入参：bigint|double|decimal
	出参：double	出参：double
VERSION	入参：なし	入参：なし
	出参：文字列	出参：文字列
WEEKDAY	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int
WEEKOFYEAR	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int
WIDTH_BUCKET	入参：(bigint|double|decimal, bigint|double|decimal, bigint|double|decimal, int)	入参：(bigint|double|decimal, bigint|double|decimal, bigint|double|decimal, int)
	出参：int	出参：int
WINDOW	入参：(date|timestamp, string[, string[, string]])	–

	出参：table	
XPATH	入参：(string, string)	入参：(string, string)
	出参：array	出参：array
XPATH_BOOLEAN	入参：(string, string)	入参：(string, string)
	出参：boolean	出参：boolean
XPATH_DOUBLE	入参：(string, string)	入参：(string, string)
	出参：double	出参：double
XPATH_FLOAT	入参：(string, string)	入参：(string, string)
	出参：float	出参：float
XPATH_INT	入参：(string, string)	入参：(string, string)
	出参：int	出参：int
XPATH_LONG	入参：(string, string)	入参：(string, string)
	出参：bigint	出参：bigint
XPATH_NUMBER	入参：(string, string)	入参：(string, string)
	出参：double	出参：double
XPATH_STRING	入参：(string, string)	入参：(string, string)
	出参：文字列	出参：文字列
XXHASH64	入参：(T, U, ...)	入参：(T, U, ...)
	出参：bigint	出参：bigint
YEAR	入参：date|timestamp|string	入参：date|timestamp|string
	出参：int	出参：int

二項関数

最終更新日：：2025-12-25 12:00:06

CRC32

- 関数構文：

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

- サポートエンジン：`SPARKSQL`，`PRESTO`
- 使用説明：CRC32アルゴリズムを使用して式の巡回冗長検査値を計算します。
- 返される型：`bigint`
- 例：

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

MD5

- 関数構文：

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

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明：MD5 128ビットチェックサムを16進数文字列形式で返します。
- 返される型：string
- 例：

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

HASH

- 関数構文：

```
HASH(&lt;expr1&gt; any[, &lt;expr2&gt; any, ...])
```

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明：すべてのパラメータのハッシュ値を返します。 `SPARKSQL` と `PRESTO` の計算方法は異なるため、異なる結果が得られる可能性があります。
- 戻り型： `integer`
- 例：

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

XXHASH64

- 関数構文：

```
XXHASH64(&lt;expr1&gt; any[, &lt;expr2&gt; any, ...])
```

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明：返されるパラメータの64ビットハッシュ値。 `SPARKSQL` と `PRESTO` の計算方法が異なるため、異なる結果が得られる可能性があります
- 返される型：bigint
- 例：

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

SHA

- 関数構文：

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

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明： `expr` のsha1ハッシュ値を16進数文字列形式で返します。
- 戻り値の型： `string`
- 例：

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

```
f94b2c96e2f127726ef4bcec6bc779f0f2e7888f
```

SHA1

- 関数構文：

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

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明： `expr` のsha1ハッシュ値を16進数文字列形式で返します。
- 戻り値の型： `string`
- 例：

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

SHA2

- 関数構文：

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

- サポートエンジン：SPARKSQL, PRESTO
- 使用説明： `expr` のSHA-2ファミリーのチェックサムを16進数文字列形式で返します。SHA-224、SHA-256、SHA-384、SHA-512をサポートしています。ビット長0は256と同等です。
- 戻り値の型： `string`
- 例：

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

AES_ENCRYPT

- 関数構文：

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

- サポートエンジン：PRESTO
- 使用説明：AESアルゴリズムを使用して `expr` を暗号化します
- 戻り値の型： `binary`
- 例：

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

AES_DECRYPT

- 関数構文：

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

- サポートエンジン： `PRESTO`
- 使用説明：AESアルゴリズムを使用して `expr` を復号化します
- 戻り値の型： `binary`
- 例：

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

ビット演算関数

最終更新日：：2025-12-25 12:00:06

BIT_COUNT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto
- 使用説明：`expr`を符号なし64ビット整数に設定し、そのビットが1の数を返します。パラメータがNULLの場合、NULLを返します。
- 戻り値の型：`integer`
- 例：

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

BIT_GET

- 関数の構文：

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

- サポートエンジン：SparkSQL、Presto
- 使用説明：指定した位置（0または1）の値を返します。位置は右から左に番号が付けられ、0から始まります。位置パラメータは負であってはなりません。
- 戻り型：`integer`
- 例：

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

GETBIT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto
- 使用説明：指定された位置（0または1）の値を返します。位置は右から左に番号が付けられ、0から始まります。位置パラメータは負の値にできません。
- 戻り値の型：`integer`
- 例：

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

集約関数

最終更新日：：2025-12-25 12:00:07

ARRAY

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：指定された要素に基づいて配列を生成します
- 戻り値の型：array<T>。
- 例：

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

FILTER

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：与えられた述語を使用して入力配列をフィルタリングします。
- 戻り値の型：array<T>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL
- 使用説明：funcを使用して配列内の要素を変換します。
- 戻り値の型：array<U>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：2つの指定された配列を要素ごとに結合して単一の配列にするには関数を使用します。一方の配列が短い場合、関数を適用する前に、長い配列の長さに合わせるために末尾にnullを追加します。
- 戻り値の型：array<R>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：配列内のすべての要素にテスト述語が適用されるかどうかを確認します。
- 戻り値の型：boolean。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：配列 expr の要素を集約します。
- 戻り値の型：R。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：述語が配列内の1つ以上の要素に適用されるかどうかをテストします。
- 戻り値の型：boolean。
- 例：

```
> 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

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：配列にvalueが含まれている場合、trueを返します。
- 戻り値の型：boolean。
- 例：

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

ARRAYS_OVERLAP

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：a が b にも存在する非 null 要素を少なくとも1つ含む場合、true を返します。配列に共通要素がなく、両方が非空で、いずれかが null 要素を含む場合は null を返し、それ以外の場合は false を返します。
- 戻り値の型：boolean。
- 例：

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

ARRAY_INTERSECT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：a と b の交差部分の要素を重複なしで配列として返します。
- 戻り値の型：array<T>。
- 例：

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

ARRAY_JOIN

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：指定された配列の要素を区切り文字とオプションの文字列で結合し、nullを置換します。nullReplacementの値が設定されていない場合、すべてのnull値はフィルタリングされます。
- 戻り値の型：string。
- 例：

```
> 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

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：配列の最初の要素の（1から数える）インデックスを返します。
- 戻り値の型：integer。
- 例：

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

ARRAY_SORT

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：入力配列をソートします。funcを省略した場合、昇順でソートされます。
- 戻り値の型：array<T>。
- 例：

```
> 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

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：a にあって b にはない要素を重複なしで配列として返します。
- 戻り値の型：array<T>。
- 例：

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

ARRAY_UNION

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：a と b の和集合の要素を重複なしで配列として返します。
- 戻り値の型：array<T>。
- 例：

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

NAMED_STRUCT

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：指定されたフィールド名と値を使用してstructを作成します。
- 戻り値の型：struct。
- 例：

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

STRUCT

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用方法：指定されたフィールド名と値を使用してstructを作成します。
- 戻り値の型：struct。
- 例：

```
> 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

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：配列aのインデックスstart（配列のインデックスは1から始まり、startが負の場合は末尾から開始）から、指定された長さlengthのサブセットを返します。
- 戻り値の型：array<T>。
- 例：

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

ARRAYS_ZIP

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：結合された配列を返します。要素はstruct型で、N番目のstructには各入力配列のN番目の値が含まれます。
- 戻り値の型：array<struct<string, T>>。
- 例：

```
> 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

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用方法：入力配列を昇順または降順でソートします。double/float型の場合、NaNは非NaN要素より大きいと見なされます。Null要素は、昇順の場合は返される配列の先頭に、降順の場合は末尾に配置されます。
- 戻り値の型：array<T>。
- 例：

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

SHUFFLE

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：指定された配列のランダムな順列を返します。
- 戻り値の型：array<T>。
- 例：

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

ARRAY_MAX

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：配列内の最大値を返します。double/float型の場合、NaNは非NaN要素より大きいと見なされます。空要素はスキップします。
- 戻り値の型：array<T>。
- 例：

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

ARRAY_MIN

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。

- 使用説明：配列内の最小値を返します。double/float型の場合、NaNは非NaN要素より大きいと見なされます。空要素はスキップします。
- 戻り値の型：array<T>。
- 例：

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

FLATTEN

- 関数構文：

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

- 対応エンジン：SparkSQL、Presto。
- 使用説明：二次元配列を一次元配列に変換します。
- 戻り値の型：array<T>。
- 例：

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

SEQUENCE

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto
- 使用説明：startからend（endを含む）までの配列を生成し、段階的に増加させます。返される要素の型はstartとendの型と同じです。
  startとstopの式は同じ型に解決されなければなりません。開始および停止式がdateまたはtimestamp型に解決される場合、stepはinterval、year-month interval、またはday-time interval型に解決されなければなりません。それ以外の場合は、startとendと同じ型に解決されます。
- 戻り値の型：`startと同じ`
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：count 個の element を含む配列を返します。
- 戻り値の型：array<T>。
- 例：

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

ARRAY_REMOVE

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：配列から指定された要素と等しいすべての要素を削除します。
- 戻り値の型：array<T>。
- 例：

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

```
 [1,2,null]
```

ARRAY_DISTINCT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：配列から重複する値を削除します。
- 戻り値の型：array<T>。
- 例：

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

ELEMENT_AT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto
- 使用説明：指定された（1から数える）インデックスの配列要素を返すか、指定されたキーの値を返します。
- 戻り値の型：T，V
- 例：

```
> SELECT element_at(array(1, 2, 3), 2);
 2
> SELECT element_at(map(1, 'a', 2, 'b'), 2);
 b
```

MAP

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された要素に基づいてマップを生成します。
- 戻り値の型：MAP<K, V>。
- 例：

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

MAP_FROM_ARRAYS

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された key/value 配列のペアを使用してマップを作成します。keys 内のすべての要素は null であってはなりません。
- 戻り値の型：map<K, V>。
- 例：

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

MAP_KEYS

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：mapキーを含む順不同の配列を返します。
- 戻り値の型：array<K>。
- 例：

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

MAP_VALUES

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：map値を含む順不同の配列を返します。
- 戻り値の型：array<V>。
- 例：

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

MAP_ENTRIES

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたmap内のすべての項目を含む順不同の配列を返します。
- 戻り値の型：array<struct<K, V>>。
- 例：

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

MAP_FROM_ENTRIES

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：与えられたエントリ配列から作成されたマップを返します。
- 戻り値の型：map<K, V>。
- 例：

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

MAP_CONCAT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたすべてのマッピングの和集合を返します。
- 戻り値の型：map<K, V>。
- 例：

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

MAP_FILTER

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：funcを使用してmのエントリをフィルタリングします。
- 戻り値の型：map<K, V>。
- 例：

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

MAP_ZIP_WITH

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：funcを使用して、2つの指定されたマッピングを単一のマッピングに結合します。1つのマッピングにのみ表示されるキーの場合、値はNULLに設定されます。入力マッピングに重複キーが含まれている場合、重複キーの最初のエントリのみがfuncに渡されます。
- 戻り値の型：MAP<K, R>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：func を使用して map のキーを変換します。
- 戻り値の型：map<R, V>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：funcを使用してmap内のvaluesを変換します。
- 戻り値の型：map<K, R>。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：配列またはマップのサイズを返します。
- 戻り値の型：integer。
- 例：

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

CARDINALITY

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：配列またはマップのサイズを返します。
- 戻り値の型：integer。
- 例：

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

ANY_MATCH / FORALL

- 関数構文：

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

- サポートエンジン：SparkSQL。
- 使用説明：配列の各要素に対して、順番にlambda式を実行し、1つでもtrueが返された場合に関数はtrueを返し、それ以外の場合はfalseを返します。
  - 配列にNULLの要素がある場合、NULLを返します。
- 戻り値の型：boolean
- 例：

```
> 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
```

日時関数

最終更新日：：2025-12-25 12:00:07

DATE

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：強制的にdate型に変換します。
- 戻り値の型：date。
- 例：

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

TIMESTAMP

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：timestamp への強制型変換。
- 戻り値の型：timestamp。
- 例：

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

ADD_MONTHS

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：start_date から num か月後の日付を返します。
- 戻り値の型：date。
- 例：

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

CURRENT_DATE

- 関数構文：

```
CURRENT_DATE
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：クエリ計算開始時の現在の日付を返します。
- 戻り値の型：date。
- 例：

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

CURRENT_TIMESTAMP

- 関数構文：

```
CURRENT_TIMESTAMP
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：クエリ計算開始時の現在のタイムスタンプを返します。
- 戻り値の型：timestamp。
- 例：

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

CURRENT_TIMEZONE

- 関数構文：

```
CURRENT_TIMEZONE(
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：現在のセッションのローカルタイムゾーンを返します。
- 戻り型：string。
- 例：

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

DATEDIFF

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：start から end までの日数を返します。
- 戻り型：integer。
- 例：

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

DATE_ADD

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：start_date から num 日後の日付を返します。
- 戻り値の型：date。

- 例：

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

DATE_FORMAT

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：タイムスタンプを指定された日付形式の文字列値に変換します。
- 戻り型：string。
- 例：

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

DATE_SUB

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：start_date から num 日前の日付を返します。
- 戻り値の型：date。
- 例：

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

DAY

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：日付/タイムスタンプdがその月の何日目かを計算します。
- 戻り型：integer。
- 例：

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

DAYOFYEAR

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：dがその年の何日目かを計算します。
- 戻り型：integer。
- 例：

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

DAYOFMONTH

- 関数構文：

```
DAYOFMONTH
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：日付/タイムスタンプdがその月の何日目かを計算します。
- 戻り型：integer。
- 例：

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

```
 30
```

FROM_UNIXTIME

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：unix_timeが表す日付/時刻を形式fmtで返します。fmtを省略した場合、'yyyy-MM-dd HH:mm:ss'が使用されます。
- 戻り型：string。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：utcタイムスタンプを指定し、その時間を指定されたタイムゾーンのタイムスタンプとして表示します。
- 戻り値の型：timestamp。
- 例：

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

HOUR

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたタイムスタンプ ts の時間部分を返します。
- 戻り型：integer。
- 例：

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

LAST_DAY

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された日付 d の現在の月の最終日を返します。
- 戻り値の型：date。
- 例：

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

MINUTE

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：タイムスタンプ ts の分を返します。
- 戻り型：integer。
- 例：

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

```
 58
```

MONTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された日付 d の月を返します。
- 戻り型：integer。
- 例：

```
> SELECT month('2016-07-30');
 7
```

MONTHS_BETWEEN

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：ts1がts2より遅い場合、結果は正になります。ts1とts2が当月の同じ日である場合、または両方が当月の最終日である場合、その日の時間は無視されます。それ以外の場合、差は月ごとに31日で計算され、roundOffがfalseでない限り、8桁に丸められます。
- 戻り型：double。
- 例：

```
> 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

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：start_date後の最初の指定曜日を返します。
- 戻り値の型：
  SparkSQL：date
  Presto：string
- 例：

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

NOW

- 関数構文：

```
NOW()
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：現在のタイムスタンプを返します。
- 戻り値の型：timestamp。
- 例：

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

QUARTER

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：d が属する四半期を返します。
- 戻り型：integer。
- 例：

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

SECOND

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：現在のタイムスタンプの秒数を返します。
- 戻り型：integer。
- 例：

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

TO_TIMESTAMP

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：fmt 形式の ts_str 式をタイムスタンプに解析します。無効な入力の場合は NULL を返します。デフォルトでは、fmt を省略すると、タイムスタンプの強制変換ルールに従います。結果のデータ型は設定値と一致します。
- 戻り値の型：timestamp。
- 例：

```
> 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

- 関数構文：

```
TO_DATE(<date_str> string[, <fmt> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：fmt 形式の date_str 式を日付に解析します。無効な入力の場合は NULL を返します。デフォルトでは、fmt を省略すると、日付の強制変換ルールに従います。結果のデータ型は設定値と一致します。
- 戻り値の型：date。
- 例：

```
> 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

- 関数構文：

```
TO_UNIX_TIMESTAMP(<ts> date|timestamp|string[, <fmt> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：tsのunixタイムスタンプを返します。
- 戻り型：bigint。
- 例：

```
> SELECT to_unix_timestamp('2016-04-08', 'yyyy-MM-dd');
 1460098800
```

TO_UTC_TIMESTAMP

- 関数構文：

```
-- SparkSQL
TO_UTC_TIMESTAMP(<ts> date|timestamp|string, <timezone> string)

-- Presto
```

```
TO_UTC_TIMESTAMP(<ts> date|timestamp|string|interger|double|decimal,
<timezone> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたタイムゾーンのタイムスタンプをUTCに変換します。
- 戻り値の型：timestamp。
- 例：

```
-- 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

- 関数構文：

```
TRUNC(<d> date|string, <fmt> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された日付 d を fmt で指定された時間単位で切り捨てた後の日付値を返します。
- 戻り値の型：date。
- 例：

```
> 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

- 関数構文：

```
DATE_TRUNC(<fmt> string, <ts> date|timestamp|string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：タイムスタンプ ts を fmt に従って切り捨てた後のタイムスタンプを返します。
- 戻り値の型：timestamp。
- 例：

```
> 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

- 関数構文：

```
UNIX_TIMESTAMP([<ts> date|timestamp|string[, fmt]])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：現在または指定された時刻のUNIXタイムスタンプを返します。
- 戻り型：bigint。
- 例：

```
> SELECT unix_timestamp();
 1476884637
> SELECT unix_timestamp('2016-04-08', 'yyyy-MM-dd');
 1460041200
```

DAYOFWEEK

- 関数構文：

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

- サポートエンジン：parkSQL、Presto。
- 使用説明：返される日付/タイムスタンプの「d」は曜日を表します。
- 戻り型：nteger。
- 例：

```
> SELECT dayofweek('2009-07-30');
 5
```

WEEKDAY

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：「d」は曜日を表す日付/タイムスタンプを返します。
- 戻り型：integer。
- 例：

```
> SELECT weekday('2009-07-30');
 3
```

WEEKOFYEAR

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された日付「d」が年間の第何週目であるかを返します。
- 戻り型：integer。
- 例：

```
> SELECT weekofyear('2008-02-20');
```

```
 8
```

YEAR

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：日付/タイムスタンプ「d」の年を返します。
- 戻り型：integer。
- 例：

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

MAKE_DATE

- 関数構文：

```
MAKE_DATE(<year> integer, <month> integer, <day> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：year、month、dayフィールドから日付を作成します。
- 戻り型：date。
- 例：

```
> SELECT make_date(2013, 7, 15);
 2013-07-15
> SELECT make_date(2019, 7, NULL);
 NULL
```

MAKE_TIMESTAMP

- 関数構文：

```
MAKE_TIMESTAMP(<year> integer, <month> integer, <day> integer, <hour>
integer, <min> integer, <sec> integer|double|decimal[, <timezone>
```

```
string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたフィールドに基づいてタイムスタンプを作成します。
- 戻り型：timestamp。
- 例：

```
> 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

- 関数構文：

```
DATE_PART(<field> string, <source> date|timestamp)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：日付/タイムスタンプの一部を抽出します。
- 戻り型：integer|double。
- 例：

```
> 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

- 関数構文：

```
DATE_FROM_UNIX_DATE(<unix_timestamp> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：1970-01-01からの日数に基づいて日付を作成します。
- 戻り値の型：date。
- 例：

```
> SELECT date_from_unix_date(1);
 1970-01-02
```

UNIX_DATE

- 関数構文：

```
UNIX_DATE(<d> date)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：1970-01-01からの日数を返します。
- 戻り型：integer。
- 例：

```
> SELECT unix_date(DATE("1970-01-02"));
 1
```

TIMESTAMP_SECONDS

- 関数構文：

```
TIMESTAMP_SECONDS(<sec> bigint|double|decimal)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：UTCエポックからの秒数でタイムスタンプを作成します。
- 戻り値の型：timestamp。

- 例：

```
> SELECT timestamp_seconds(1230219000);
 2008-12-25 07:30:00
> SELECT timestamp_seconds(1230219000.123);
 2008-12-25 07:30:00.123
```

TIMESTAMP_MILLIS

- 関数構文：

```
TIMESTAMP_MILLIS(<milli> bigint|double|decimal)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：UTCエポックからのミリ秒数でタイムスタンプを作成します。
- 戻り値の型：timestamp。
- 例：

```
> SELECT timestamp_millis(1230219000123);
 2008-12-25 07:30:00.123
```

TIMESTAMP_MICROS

- 関数構文：

```
TIMESTAMP_MICROS(<micro> bigint)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：UTCエポックからのミリ秒数でタイムスタンプを作成します。
- 戻り値の型：timestamp。
- 例：

```
> SELECT timestamp_micros(1230219000123123);
 2008-12-25 07:30:00.123123
```

UNIX_SECONDS

- 関数構文：

```
UNIX_SECONDS(<ts> timestamp)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：1970-01-01 00:00:00 UTCからの秒数を返します。
- 戻り型：bigint。
- 例：

```
> SELECT unix_seconds(TIMESTAMP('1970-01-01 00:00:01Z'));
 1
```

UNIX_MILLIS

- 関数構文：

```
UNIX_MILLIS(<ts> timestamp)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：1970-01-01 00:00:00 UTCからのミリ秒数を返します。
- 戻り型：bigint。
- 例：

```
> SELECT unix_millis(TIMESTAMP('1970-01-01 00:00:01Z'));
 1000
```

UNIX_MICROS

- 関数構文：

```
UNIX_MICROS(<ts> timestamp)
```

- サポートエンジン：SparkSQL。
- 使用説明：1970-01-01 00:00:00 UTCからのマイクロ秒数を返します。
- 戻り型：bigint。
- 例：

```
> SELECT unix_micros(timestamp '1970 00:00:00.001')
```

```
1000
```

DATEADD/TIMESTAMP_ADD

- 関数構文：

```
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
TIMESTAMP_ADD(<date|timestamp> dt, <int> delta, <string> pattern)
```

- サポートエンジン：SparkSQL。
- エンジンバージョン要件：2023年12月07日以降に購入したエンジン、または最新のエンジンにアップグレードする必要があります。
- 使用説明：指定された日付または時刻に、指定された単位とオフセットに基づいて最終的な時間を計算します。
  - dt：入力された日付または時刻の形式。nullの場合はnullを返します。
  - delta：必須、intタイプ。nullの場合はnullを返します。
  - pattern：必須、stringタイプ、単位。patternは以下をサポートできます：
  - 年（標準「y」、互換性あり「-year」、「yyyy」）
  - 月（標準「M」、互換性あり「-month」、「-mon」、「MM」）
  - 日（標準「d」、互換性あり「-day」、「dd」）
  - 時間（標準「H」、互換性のある形式：「-hour」、「hh」、「HH」、「h」）
  - 分（標準「m」、互換性のある「mi」、「mm」）
  - 秒（標準「s」、「ss」にも対応）
  - マイクロ秒（標準「S」、互換性のある「SSS」）

  他のタイプの入力は許可されていません。そうでない場合はエラーが発生します。

> ⚠ **注意：**
> - デルタの単位が月の場合、デルタ値を加算した後の月部分がDayのオーバーフローを引き起こさない場合、Dayの値は変更されません。それ以外の場合、Dayの値は結果の月の最終日に設定されます。
> - 出力時間はデフォルトでタイムゾーンとなります。
> - 月Mと分mの単位の区別；0-12時間制hと0-24時間制Hに対応していますが、Hの使用を推奨します。

- 戻り値の型：timestamp。
- 例：

```
> 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 関数

最終更新日：：2025-12-25 12:00:07

GET_JSON_OBJECT

- 関数構文：

```
GET_JSON_OBJECT(<json> string, <path> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：jsonオブジェクトを抽出します。
- 戻り値の型：string。
- 例：

```
> SELECT get_json_object('{"a":"b"}', '$.a');
 b
```

JSON_TUPLE

- 関数構文：

```
JSON_TUPLE(<json> string, <p1> string, ..., <pn> string)
```

- サポートエンジン：SparkSQL。
- 使用説明：get_json_object関数に似たタプルを返しますが、複数の名前を入力する必要があります。すべての入力パラメータと出力列の型は文字列です。
- 戻り値の型：struct<string, ..., string>。
- 例：

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

TO_JSON

- 関数構文：

```
TO_JSON(<expr> struct|map|array[, <option> map])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された構造値を持つJSON文字列を返します。
- 「設定–一般–クイックフローティングウィンドウ」で自動表示をオンにできます
- 例：

```
> 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

- 関数構文：

```
FROM_JSON(<json> string, <schema> string[, <options> map<string,
string>])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定されたjsonStrとスキーマを持つ構造値を返します。
- 戻り値の型：struct。
- 例：

```
> 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

- 関数構文：

```
SCHEMA_OF_JSON(<json> string[, <options> map<string, string>])
```

- サポートエンジン：SparkSQL。
- 使用説明：JSON文字列のDDL形式の構造を返します。
- 戻り値の型：string。
- 例：

```
> 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

- 関数構文：

```
JSON_ARRAY_LENGTH(<jsonArray> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：最外層のJSON配列内の要素数を返します。
- 戻り値の型：integer。
- 例：

```
> 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

- 関数構文：

```
JSON_OBJECT_KEYS(<json> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：最外層のJSONオブジェクトのすべてのキーを配列形式で返します。
- 戻り値の型：array<string>。
- 例：

```
> SELECT json_object_keys('{}');
[]
> SELECT json_object_keys('{"key": "value"}');
["key"]
> SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}');
["f1","f2"]
```

数学関数

最終更新日：：2025-12-25 12:00:07

ABS

- 関数構文：

```
ABS(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの絶対値を返します
- 戻り型：<expr>と一致
- 例：

```
> SELECT abs(-1);
 1
```

ACOS

- 関数構文：

```
ACOS(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの逆余弦値を返します
- 戻り型：double
- 例：

```
> SELECT acos(1);
 0.0
> SELECT acos(2);
 NaN
```

ACOSH

- 関数構文：

```
ACOSH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの逆双曲余弦値を返します
- 戻り型：double
- 例：

```
> SELECT acosh(1);
 0.0
> SELECT acosh(0);
 NaN
```

ASIN

- 関数構文：

```
ASIN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの逆正弦値を返します
- 戻り型：double
- 例：

```
> SELECT asin(0);
 0.0
> SELECT asin(2);
 NaN
```

ASINH

- 関数構文：

```
ASINH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL
- 使用説明：exprの逆双曲正弦値を返します
- 戻り型：double

- 例：

```
> SELECT asinh(0);
 0.0
```

ATAN

- 関数構文：

```
ATAN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの逆正接値を返します
- 戻り型：double
- 例：

```
> SELECT atan(0);
 0.0
```

ATAN2

- 関数構文：

```
ATAN2(<x>, integer|double|decimal, <y> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：平面の正x軸と指定された座標点間のラジアン角を返します
- 戻り型：double
- 例：

```
> SELECT atan2(0, 0);
 0.0
```

ATANH

- 関数構文：

```
ATANH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの逆双曲正接値を返します
- 戻り型：double
- 例：

```
> SELECT atanh(0);
 0.0
> SELECT atanh(2);
 NaN
```

BIN

- 関数構文：

```
BIN(<expr> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：長値exprのバイナリ表現の文字列表現を返します。
- 戻り型：string
- 例：

```
> SELECT bin(13);
 1101
> SELECT bin(-13);
 1111111111111111111111111111111111111111111111111111111111110011
```

BROUND

- 関数構文：

```
BROUND(<expr> integer|double|decimal, <d> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprをHALF_EVEN丸めモードでd桁の小数に丸めて返します
- 戻り型：decimal

- 例：

```
> SELECT bround(2.5, 0);
 2
```

CBRT

- 関数構文：

```
CBRT(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの立方根を返します
- 戻り型：double
- 例：

```
> SELECT cbrt(27.0);
 3.0
```

CEIL

- 関数構文：

```
CEIL(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr以上の最小の整数を返します
- 戻り型：integer
- 例：

```
> SELECT ceil(-0.1);
 0
> SELECT ceil(5);
 5
```

COS

- 関数構文：

```
COS(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの余弦値を返します
- 戻り型：double
- 例：

```
> SELECT cos(0);
 1.0
```

COSH

- 関数構文：

```
COSH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの双曲余弦値を返します
- 戻り型：double
- 例：

```
> SELECT cosh(0);
 1.0
```

CONV

- 関数構文：

```
CONV(<num> bigint|string, <from_base> integer, <to_base> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：numをfrom_baseからto_baseに変換します
- 戻り型：string
- 例：

```
> SELECT conv('100', 2, 10);
```

```
 4
> SELECT conv(-10, 16, -10);
 -16
```

DEGREES

- 関数構文：

```
DEGREES(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ラジアンを度に変換する
- 戻り型：double
- 例：

```
> SELECT degrees(3.141592653589793);
 180.0
```

E

- 関数構文：

```
E()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：オイラー定数を返します
- 戻り型：double
- 例：

```
> SELECT e();
 2.718281828459045
```

EXP

- 関数構文：

```
EXP(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：オイラー定数eのexpr乗を返します
- 戻り型：double
- 例：

```
> SELECT exp(0);
 1.0
```

EXPM1

- 関数構文：

```
EXPM1(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL
- 使用説明：EXP(expr)–1を返します
- 戻り型：double
- 例：

```
> SELECT expm1(0);
 0.0
```

FLOOR

- 関数構文：

```
FLOOR(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr以下の最大の整数を返します
- 戻り型：integer
- 例：

```
> SELECT floor(-0.1);
 -1
> SELECT floor(5);
 5
```

FACTORIAL

- 関数構文：

```
FACTORIAL(<expr> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの階乗を返します。exprは[0..20]の範囲です。それ以外の場合はNULLを返します。
- 戻り型：bigint
- 例：

```
> SELECT factorial(5);
 120
```

HEX

- 関数構文：

```
HEX(<expr> bigint|string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprを16進数で返します
- 戻り型：string
- 例：

```
> SELECT hex(17);
 11
> SELECT hex('Spark SQL');
 537061726B2053514C
```

HYPOT

- 関数構文：

```
HYPOT(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：sqrt(pow(expr1, 2) + pow(expr2, 2))を返します

- 戻り値の型：double
- 例：

```
> SELECT hypot(3, 4);
 5.0
```

LOG

- 関数構文：

```
LOG(<base> integer|double|decimal, <expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprのbaseを底とする対数を返します。
- 戻り値の型：double
- 例：

```
> SELECT log(10, 100);
 2.0
```

LOG10

- 関数構文：

```
LOG10(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの10を底とする対数を返します。
- 戻り値の型：double
- 例：

```
> SELECT log10(10);
 1.0
```

LOG1P

- 関数構文：

```
LOG1P(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：log(1 + expr)を返します
- 戻り値の型：double
- 例：

```
> SELECT log1p(0);
 0.0
```

LOG2

- 関数構文：

```
LOG2(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの2を底とする対数を返します。
- 戻り値の型：double
- 例：

```
> SELECT log2(2);
 1.0
```

LN

- 関数構文：

```
LN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの自然対数（eを底とする）を返します。
- 戻り値の型：double
- 例：

```
> SELECT ln(1);
```

```
 0.0
```

MOD

- 関数構文：

```
MOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
<expr1> MOD <expr2>
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1/expr2の余りを返します。
- 戻り型：double|integer
- 例：

```
> SELECT 2 % 1.8;
 0.2
> SELECT MOD(2, 1.8);
 0.2
```

NEGATIVE

- 関数構文：

```
NEGATIVE(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの負数を返します
- 戻り型：exprと一致
- 例：

```
> SELECT negative(1);
 -1
```

PI

- 関数構文：

```
PI()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：PIを返す
- 戻り値の型：double
- 例：

```
> SELECT pi();
 3.141592653589793
```

PMOD

- 関数構文：

```
PMOD(<expr1> integer|double|decimal, <expr2> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1 mod expr2の正の値を返します
- 戻り値の型：double
- 例：

```
> SELECT pmod(10, 3);
 1
> SELECT pmod(-10, 3);
 2
```

POSITIVE

- 関数構文：

```
POSITIVE(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprを返します
- 戻り型：exprと一致
- 例：

```
> SELECT positive(1);
 1
```

POWER

- 関数構文：

```
POWER(<base> integer|double|decimal, <number> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：baseのnumber乗を返します。
- 戻り値の型：double
- 例：

```
> SELECT power(2, 3);
 8.0
```

POW

- 関数構文：

```
POW(<base> integer|double|decimal, <number> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：baseのnumber乗を返します
- 戻り型：double
- 例：

```
> SELECT pow(2, 3);
 8.0
```

RADIANS

- 関数構文：

```
RADIANS(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：度をラジアンに変換する
- 戻り型：double
- 例：

```
> SELECT radians(180);
 3.141592653589793
```

RINT

- 関数構文：

```
RINT(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：パラメータ値に最も近く、数学的な整数に等しい倍精度値を返します
- 戻り型：double
- 例：

```
> SELECT rint(12.3456);
 12.0
```

ROUND

- 関数構文：

```
ROUND(<expr> integer|double|decimal, <d> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprを半切り上げモードでd桁の小数に丸めます。
- 戻り型：double
- 例：

```
> SELECT round(2.5, 0);
 3
```

SHIFTLEFT

- 関数構文：

```
SHIFTLEFT(<base> integer|double|decimal, <expr> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ビット単位で左にシフトします。
- 戻り型：int |bigint
- 例：

```
> SELECT shiftleft(2, 1);
 4
```

SHIFTRIGHT

- 関数構文：

```
SHIFTRIGHT(<base> integer|double|decimal, <expr> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ビット単位で右にシフトします。
- 戻り型：int |bigint
- 例：

```
> SELECT shiftright(4, 1);
 2
```

SHIFTRIGHTUNSIGNED

- 関数構文：

```
SHIFTRIGHTUNSIGNED(<base> integer|double|decimal, <expr> integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ビット単位で符号なし右シフトします。
- 戻り型：int |bigint
- 例：

```
> SELECT shiftrightunsigned(4, 1);
 2
```

SIGN

- 関数構文：

```
SIGN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprが負、0、または正の場合、それぞれ–1.0、0.0、または1.0を返します。
- 戻り型：double
- 例：

```
> SELECT sign(40);
 1.0
```

SIGNUM

- 関数構文：

```
SIGNUM(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprが負、0、または正の場合、それぞれ–1.0、0.0、または1.0を返します。
- 戻り型：double
- 例：

```
> SELECT signum(40);
 1.0
```

SIN

- 関数構文：

```
SIN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの正弦値を返します
- 戻り型：double
- 例：

```
> SELECT sin(0);
 0.0
```

SINH

- 関数構文：

```
SINH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの双曲正弦値を返します
- 戻り型：double
- 例：

```
> SELECT sinh(0);
 0.0
```

SQRT

- 関数構文：

```
SQRT(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの平方根を返します
- 戻り型：double
- 例：

```
> SELECT sqrt(4);
 2.0
```

TAN

- 関数構文：

```
TAN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの正接値を返します
- 戻り値の型：double
- 例：

```
> SELECT tan(0);
 0.0
```

COT

- 関数構文：

```
COT(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：式の余接を返します
- 戻り値の型：double
- 例：

```
> SELECT cot(1);
 0.6420926159343306
```

TANH

- 関数構文：

```
TANH(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの双曲正接値を返します
- 戻り値の型：double
- 例：

```
> SELECT tanh(0);
 0.0
```

WIDTH_BUCKET

- 関数構文：

```
WIDTH_BUCKET(<value> integer|double|decimal, <min>
integer|double|decimal, <max> integer|double|decimal, <num_backet>
integer)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：minからmaxをnum_backet個のグループに等分割し、valueが属するグループ番号を返します
- 戻り型：integer
- 例：

```
> 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

- 関数構文：

```
TRY_ADD(<expr1> integer|double|decimal|date|timestamp, <expr2>
integer|double|decimal|date|timestamp)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1とexpr2の合計を返します。オーバーフロー時は結果がnullになります。
- 戻り型：integer|double|decimal|date|timestamp
- 例：

```
> 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

- 関数構文：

```
TRY_DIVIDE(<dividend> integer|double|decimal, <divisor>
integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：dividend/divisorを返します
- 戻り値の型：double
- 例：

```
> SELECT try_divide(3, 2);
 1.5
> SELECT try_divide(2L, 2L);
 1.0
> SELECT try_divide(1, 0);
 NULL
```

RAND

- 関数構文：

```
RAND([<seed> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：[0，1]の範囲で独立かつ均一に分布するランダムな値を返します。
- 戻り値の型：double
- 例：

```
> SELECT rand();
 0.9629742951434543
> SELECT rand(0);
 0.7604953758285915
> SELECT rand(null);
 0.7604953758285915
```

RANDOM

- 関数構文：

```
RANDOM([<seed> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：[0，1]の範囲で独立かつ均一に分布するランダムな値を返します。
- 戻り値の型：double
- 例：

```
> SELECT rand();
 0.9629742951434543
> SELECT rand(0);
 0.7604953758285915
> SELECT rand(null);
 0.7604953758285915
```

RANDN

- 関数構文：

```
RANDN([<seed> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：標準正規分布から抽出された独立かつ同一分布（i.i.d.）の値を含むランダム値を返します。この関数はSPARKSQLとPRESTOで実装方法が異なるため、同じseedを使用しても異なる結果が得られる場合があります。
- 戻り値の型：double
- 例：

```
> SELECT randn();
 -0.3254147983080288
> SELECT randn(0);
 1.6034991609278433
> SELECT randn(null);
 1.6034991609278433
```

DIV

- 関数構文：

```
<expr1> DIV <expr2>
```

- サポートエンジン：SparkSQL
- 使用説明：expr1をexpr2で割ります。
- 戻り型：integer
- 例：

```
> SELECT 3 div 2;
 1
```

文字列関数

最終更新日：：2025-12-25 12:00:07

ASCII

- 関数構文：

```
ASCII(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：strの最初の文字の数値を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT ascii('222');
 50
```

BASE64

- 関数構文：

```
-- SparkSQL
BASE64(<str> string|binary)
-- Presto
BASE64(<str> binary)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：パラメータをbase64文字列に変換します。
- 戻り型：string。
- 例：

```
> SELECT base64('tencent');
 dGVuY2VudA==
```

BIT_LENGTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列データのビット長またはバイナリデータのビット数を返します。
- 戻り値の型：integer。
- 例：

```
> select bit_length('tencent');
 56
> select bit_length(binary('tencent'));
 56
```

CHAR

- 関数構文：

```
CHAR(<expr> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：expr の ASCII 文字を返します。expr が 256 より大きい場合、expr=expr%256 となります。
- 戻り型：string。
- 例：

```
> SELECT char(65);
 A
```

CHR

- 関数構文：

```
CHR(<expr> integer)
```

- サポートエンジン：SparkSQL。
- 使用説明：expr の ASCII 文字を返します。expr が 256 より大きい場合、expr=expr%256 となります。
- 戻り型：string。
- 例：

```
> SELECT chr(65);
 A
```

CHAR_LENGTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列データの文字数またはバイナリデータのバイト数を返します。文字列データの長さには末尾のスペースが含まれます。バイナリデータの長さにはバイナリゼロが含まれます。
- 戻り値の型：integer。
- 例：

```
> select char_length(binary('tencent'));
 7
> select char_length('tencent');
 7
```

CHARACTER_LENGTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列データの文字数またはバイナリデータのバイト数を返します。文字列データの長さには末尾のスペースが含まれます。バイナリデータの長さにはバイナリゼロが含まれます。
- 戻り値の型：integer。
- 例：

```
> select character_length(binary('tencent'));
 7
> select character_length('tencent');
 7
```

CONCAT_WS

- 関数構文：

```
CONCAT_WS(<sep> string[, <s> string|array<string>}]+)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：sep で区切られた文字列を返します。
- 戻り型：string。
- 例：

```
> SELECT concat_ws(' ', 'tencent', 'dlc');
tencent dlc
```

DECODE

- 関数構文：

```
DECODE(<expr> binary|string, <charset>string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：最初のパラメータを2番目のパラメータ文字セットでデコードします。
- 戻り型：string。
- 例：

```
> SELECT decode(encode('abc', 'utf-8'), 'utf-8');
 abc
```

ELT

- 関数構文：

```
ELT(<n> integer, <s1> string, <s2> string, ...)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：n番目の要素を返します。
- 戻り型：string。

- 例：

```
> SELECT elt(1, 'scala', 'java');
 scala
```

ENCODE

- 関数構文：

```
ENCODE(<expr> binary|string, <charset> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：2番目のパラメータの文字セットを使用して、最初のパラメータをエンコードします。
- 戻り型：string。
- 例：

```
> SELECT encode('abc', 'utf-8');
 abc
> SELECT encode(x'616263', 'utf-8');
 abc
```

FIND_IN_SET

- 関数構文：

```
FIND_IN_SET(<str> string, <str_array> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：カンマ区切りのリスト atr_array で指定された文字列 str のインデックス（1から開始）を返します。文字列が見つからない場合、または str にカンマが含まれている場合は0を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT find_in_set('ab','abc,b,ab,c,def');
 3
```

FORMAT_NUMBER

- 関数構文：

```
FORMAT_NUMBER(<expr1> integer|double|decimal, <expr2> string|integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：expr1の形式を「#,####,####.##」に設定し、expr2の小数点以下桁数で四捨五入します。expr2が0の場合、結果には小数点または小数部分が含まれません。
- 戻り型：string。
- 例：

```
> SELECT format_number(12332.123456, 4);
 12,332.1235
> SELECT format_number(12332.123456, '##################.###');
 12332.123
```

FORMAT_STRING

- 関数構文：

```
FORMAT_STRING(<str> string, obj <T>, ...)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：printf スタイルのフォーマット文字列内のフォーマット文字列を返します。
- 戻り型：string。
- 例：

```
> SELECT format_string("Hello World %d %s", 100, "days");
 Hello World 100 days
```

INITCAP

- 関数構文：

```
INITCAP(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：各単語の最初の文字を大文字に変更し、他のすべての文字を小文字にします。

- 戻り型：string。
- 例：

```
> SELECT initcap('sPark sql');
 Spark Sql
```

INSTR

- 関数構文：

```
INSTR(<str> string, <substr> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str内で最初にsubstrが出現するインデックス（1からカウント）を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT instr('SparkSQL', 'SQL');
 6
```

LCASE

- 関数構文：

```
LCASE(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str を小文字に変更します。
- 戻り型：string。
- 例：

```
> SELECT lcase('SparkSQL');
 sparksql
```

LENGTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列データの文字数またはバイナリデータのバイト数を返します。文字列データの長さには最後のスペースが含まれます。バイナリデータの長さにはバイナリゼロが含まれます。
- 戻り値の型：integer。
- 例：

```
> SELECT length('Spark SQL ');
 10
```

LEVENSHTEIN

- 関数構文：

```
LEVENSHTEIN(<s1> string, <s2> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：与えられた2つの文字列間のレーベンシュタイン距離を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT levenshtein('kitten', 'sitting');
 3
```

LIKE

- 関数構文：

```
LIKE(<s1> str, <s2> pattern)
<str> like <pattern>[ ESCAPE <escape>]
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：strがエスケープされたescapeを含むpatternと一致する場合、trueを返します。いずれかのパラメータがnullの場合、nullを返します。それ以外の場合はfalseを返します。
- 戻り値の型：boolean。
- 例：

```
> 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

- 関数構文：

```
LOWER(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：すべての文字を小文字に変更した str を返します。
- 戻り型：string。
- 例：

```
> SELECT lower('TENCENT');
 tencent
```

LOCATE

- 関数構文：

```
LOCATE(<substr> string, <str> string[, <pos> integer])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str内のpos位以降で最初にsubstrが出現する位置を返します。
- 戻り値の型：integer。

- 例：

```
> SELECT locate('bar', 'foobarbar');
 4
> SELECT locate('bar', 'foobarbar', 5);
 7
```

OCTET_LENGTH

- 関数構文：

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

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列データのバイト長またはバイナリデータのバイト数を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT octet_length('Spark SQL');
 9
```

LPAD

- 関数構文：

```
LPAD(<str> string, <len> integer[, <pad> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：strを返し、左側をpadで長さlenまで埋めます。strがlenより長い場合、戻り値はlen文字に切り詰められます。padが指定されていない場合、strは空白文字で埋められます。
- 戻り型：string。
- 例：

```
> SELECT lpad('hi', 5, '??');
 ???hi
> SELECT lpad('hi', 1, '??');
 h
> SELECT lpad('hi', 5);
```

```
  hi
```

LTRIM

- 関数構文：

```
LTRIM(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str から先頭の空白文字を削除します。
- 戻り型：string。
- 例：

```
> SELECT ltrim('    SparkSQL   ');
 SparkSQL
```

PARSE_URL

- 関数構文：

```
PARSE_URL(<url> string, <path> string[, <key> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：urlからパスを抽出します。
- 戻り型：string。
- 例：

```
> 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

- 関数構文：

```
POSITION(<substr> string, <str> string[, <pos> integer])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str内のpos位以降で最初にsubstrが出現する位置を返します。
- 戻り値の型：integer。
- 例：

```
> SELECT position('bar', 'foobarbar');
 4
> SELECT position('bar', 'foobarbar', 5);
 7
> SELECT POSITION('bar' IN 'foobarbar');
 4
```

PRINTF

- 関数構文：

```
PRINTF(<str> string, obj <T>, ...)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：printf スタイルのフォーマット文字列からフォーマットされた文字列を返します。
- 戻り型：string。
- 例：

```
> SELECT printf("Hello World %d %s", 100, "days");
 Hello World 100 days
```

REPEAT

- 関数構文：

```
REPEAT(<str> string, <n> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：指定された文字列をn回繰り返した文字列を返します。
- 戻り型：string。

- 例：

```
> SELECT repeat('123', 2);
 123123
```

REPLACE

- 関数構文：

```
REPLACE(<str> string, <search> string[, <replace> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str 内のすべての search を replace に置き換えます。
- 戻り型：string。
- 例：

```
> SELECT replace('ABCabc', 'abc', 'DEF');
 ABCDEF
```

OVERLAY

- 関数構文：

```
OVERLAY(<input> string, <replace> string, <pos> integer[, <len>
integer])
```

- サポートエンジン：SparkSQL。
- 使用説明：inputを、posから始まる長さlenのreplaceに置き換えます。
- 戻り値の型：string。
- 例：

```
> 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 'tructured' 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('tructured', 'utf-8') FROM 2 FOR 4);
 Structured SQL
```

RPAD

- 関数構文：

```
RPAD(<str> string, <len> integer[, <pad> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：strを返し、右側をpadで長さlenまで埋めます。strがlenより長い場合、戻り値はlen文字に切り詰められます。padが指定されていない場合、strは空白文字で埋められます。
- 戻り値の型：string。
- 例：

```
> SELECT rpad('hi', 5, '??');
 hi???
> SELECT rpad('hi', 1, '??');
 h
> SELECT rpad('hi', 5);
 hi
```

RTRIM

- 関数構文：

```
RTRIM(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str から末尾の空白文字を削除します。
- 戻り値の型：string。
- 例：

```
> SELECT rtrim('    SparkSQL   ');
   SparkSQL
```

SENTENCES

- 関数構文：

```
SENTENCES(<str> string[, <lang> string, <country> string])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：str を単語の配列に分割します。
- 戻り型：array <string>。
- 例：

```
> SELECT sentences('Hi there! Good morning.');
 [["Hi","there"],["Good","morning"]]
```

SOUNDEX

- 関数構文：

```
SOUNDEX(<str> string)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列のSoundexエンコードを返します。
- 戻り値の型：string。
- 例：

```
> SELECT soundex('Miller');
 M460
```

SPACE

- 関数構文：

```
SPACE(<n> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：n個のスペースで構成される文字列を返します。
- 戻り値の型：string。
- 例：

```
> SELECT concat(space(2), '1');
 1
```

SPLIT

- 関数構文：

```
SPLIT(<str> string, <regex> string, <limit> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：regexに一致する文字列を区切り文字として、strを分割し、最大長がlimitの配列を返します。
- 戻り型：array <string>。
- 例：

```
> SELECT split('oneAtwoBthreeC', '[ABC]');
 ["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', -1);
 ["one","two","three",""]
> SELECT split('oneAtwoBthreeC', '[ABC]', 2);
 ["one","twoBthreeC"]
```

SUBSTRING

- 関数構文：

```
SUBSTRING(<str> string, <pos> integer[, <len> integer])
SUBSTRING(<str> FROM <pos>[ FOR <len>])
```

- サポートエンジン：SparkSQL、Presto。

- 使用説明：posから始まり、長さがlenのstr部分文字列、またはposから始まり、長さがlenのバイト配列スライスを返します。
- 戻り値の型：string。
- 例：

```
> 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

- 関数構文：

```
SUBSTR(<str> string, <pos> integer[, <len> integer])
SUBSTR(<str> FROM <pos>[ FOR <len>])
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：posから始まり、長さがlenのstr部分文字列、またはposから始まり、長さがlenのバイト配列スライスを返します。
- 戻り値の型：string。
- 例：

```
> 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

- 関数構文：

```
LEFT(<str> string, <len> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列strの左端からlen文字を返します。lenが0以下の場合、結果は空の文字列になります。
- 戻り値の型：string。
- 例：

```
> SELECT left('tencent', 3);
 ten
```

RIGHT

- 関数構文：

```
RIGHT(<str> string, <len> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：文字列strの左端からlen文字を返します。lenが0以下の場合、結果は空の文字列になります。
- 戻り値の型：string。
- 例：

```
> SELECT left('tencent', 3);
 ten
```

SUBSTRING_INDEX

- 関数構文：

```
SUBSTRING_INDEX(<str> string, <delim> string, <count> integer)
```

- サポートエンジン：SparkSQL、Presto。
- 使用説明：delimがcount回出現する前に、strから部分文字列を返します。countが正数の場合、最後の区切り文字の左側のすべてを返します（左から数えて）。countが負数の場合、最後の区切り文字の右側のすべてを返します（右から数えて）。この関数はdelimのマッチング時に大文字と小文字を区別します。
- 戻り値の型：string。
- 例：

```
> SELECT substring_index('cloud.tencent.com', '.', 2);
 cloud.tencent
```

TRANSLATE

- 関数構文：

```
TRANSLATE(<input> string, <from> string, <to> string)
```

- サポートエンジン：SparkSQL。
- 使用説明：from文字列の文字をto文字列の対応する文字に置き換えて、input文字列を変換します。
- 戻り型：string。
- 例：

```
> SELECT translate('AaBbCc', 'abc', '123');
 A1B2C3
```

TRIM

- 関数構文：

```
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)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：
  trim(str) – strから先頭と末尾の空白文字を削除します。
  trim(BOTH FROM str)： strから先頭と末尾の空白文字を削除します。
  trim(LEADING FROM str)： strから先頭の空白文字を削除します。
  trim(TRAILING FROM str)： strから末尾の空白文字を削除します。
  trim(trimStr FROM str)： strから先頭と末尾のtrimStr文字を削除します。
  trim(BOTH trimStr FROM str)： strから先頭と末尾のtrimStr文字を削除します。
  trim(LEADING trimStr FROM str)： strから先頭のtrimStr文字を削除します。
  trim(TRAILING trimStr FROM str)：strから末尾のtrimStr文字を削除します。
- 戻り型：string。
- 例：

```
> 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

- 関数構文：

```
BTRIM(<str> string[, <trimStr> string])
```

- サポートエンジン：SparkSQL、Presto

- 使用説明：strの先頭と末尾からtrimStr（デフォルトはスペース）文字を削除します。
- 戻り型：string。
- 例：

```
> 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

- 関数構文：

```
UCASE(<str> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての文字を大文字に変更したstrを返します。
- 戻り型：string。
- 例：

```
> SELECT ucase('SparkSQL');
 SPARKSQL
```

UNBASE64

- 関数構文：

```
UNBASE64(<str> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：strをbase64文字列からバイナリに変換します。
- 戻り型：binary
- 例：

```
> SELECT unbase64('U3BhcmsgU1FM');
 Spark SQL
```

UNHEX

- 関数構文：

```
UNHEX(<str> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：16進数のstrをバイナリに変換します。
- 戻り型：binary
- 例：

```
> select unhex('74656E63656E74');
 tencent
```

UPPER

- 関数構文：

```
UPPER(<str> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての文字を大文字に変更したstrを返します。
- 戻り型：string。
- 例：

```
> SELECT upper('tencent');
TENCENT
```

UUID

- 関数構文：

```
UUID()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：36文字のUUIDを返します
- 戻り型：string。
- 例：

```
> SELECT uuid();
 46707d92-02f4-4817-8116-a4c3b23e6266
```

XPATH

- 関数構文：

```
XPATH(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：XPath式に一致する文字列配列をxmlノードから返します。
- 戻り型：array <string>。
- 例：

```
> 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

- 関数構文：

```
XPATH_BOOLEAN(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：XPath式の評価結果がtrueの場合、または一致するノードが見つかった場合、trueを返します。
- 戻り型：boolean
- 例：

```
> SELECT xpath_boolean('<a><b>1</b></a>','a/b');
 true
```

XPATH_DOUBLE

- 関数構文：

```
XPATH_DOUBLE(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：double型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：double
- 例：

```
> SELECT xpath_double('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_NUMBER

- 関数構文：

```
XPATH_NUMBER(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：double型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：double
- 例：

```
> SELECT xpath_number('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_FLOAT

- 関数構文：

```
XPATH_FLOAT(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto

- 使用説明：float型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：float
- 例：

```
> SELECT xpath_float('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3.0
```

XPATH_INT

- 関数構文：

```
XPATH_INT(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：int型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：integer
- 例：

```
> SELECT xpath_int('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_LONG

- 関数構文：

```
XPATH_LONG(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：bigint型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：bigint
- 例：

```
> SELECT xpath_long('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_SHORT

- 関数構文：

```
XPATH_SHORT(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：short型の値を返します。一致する項目が見つからない場合はゼロを返し、一致する項目が見つかったがその値が数値でない場合はNaNを返します。
- 戻り型：short。
- 例：

```
> SELECT xpath_short('<a><b>1</b><b>2</b></a>', 'sum(a/b)');
 3
```

XPATH_STRING

- 関数構文：

```
XPATH_STRING(<xml> string, <xpath> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：XPath式に一致する最初のxmlノードのテキスト内容を返します。
- 戻り型：string。
- 例：

```
> SELECT xpath_string('<a><b>b</b><c>cc</c></a>','a/c');
 cc
```

REGEXP_EXTRACT

- 関数構文：

```
REGEXP_EXTRACT(<str> string, <regexp> string[, <idx> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：str内でregexp式に一致し、regexグループインデックスidxに対応する最初の文字列を抽出します。

- 戻り型：string。
- 例：

```
> SELECT regexp_extract('100-200', '(\\d+)-(\\d+)', 1);
 100
```

REGEXP_EXTRACT_ALL

- 関数構文：

```
REGEXP_EXTRACT_ALL(<str> string, <regexp> string[, <idx> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：str内でregexp式に一致し、regexグループインデックスに対応するすべての文字列を抽出します。
- 戻り型：array <string>。
- 例：

```
> SELECT regexp_extract_all('100-200, 300-400', '(\\d+)-(\\d+)', 1);
 ["100","300"]
```

REGEXP_REPLACE

- 関数構文：

```
REGEXP_REPLACE(<str> string, <regexp> string, <rep> string[,
<position> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：str内でregexpに一致するすべての部分文字列をrepで置き換えます。
- 戻り型：string。
- 例：

```
> SELECT regexp_replace('100-200', '(\\d+)', 'num');
 num-num
```

REGEXP_LIKE

- 関数構文：

```
REGEXP_LIKE(<str> string, <regexp> string)
```

- サポートエンジン：SparkSQL
- 使用説明：文字列が正規表現に一致する場合は true を返し、そうでない場合は false を返します。
- 戻り型：boolean
- 例：

```
> SELECT regexp_like('%SystemDrive%\Users\John',
'%SystemDrive%\Users.*');
 true
```

REGEXP

- 関数構文：

```
REGEXP(<str> string, <regexp> string)
```

- サポートエンジン：SparkSQL
- 使用説明：文字列が正規表現に一致する場合は true を返し、そうでない場合は false を返します。
- 戻り型：boolean
- 例：

```
> SELECT regexp('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

CONCAT

- 関数構文：

```
CONCAT(<s1> string, <s2> string, ...)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：s1、s2、...に接続します
- 戻り型：string。
- 例：

```
> SELECT concat('Spark', 'SQL');
 SparkSQL
```

STR_TO_MAP

- 関数構文：

```
str_to_map(<text> string[, <pairDelim> string[, <keyValueDelim>
string]])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：区切り文字を使用してtextをキー/値ペアに分割し、mapを作成します。pairDelimのデフォルト区切り文字は「,」、keyValueDelimのデフォルト区切り文字は「:」です。pairDelimとkeyValueDelimはどちらも正規表現として扱われます。
- 戻り型：map <string, string>
- 例：

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

REVERSE

- 関数構文：

```
REVERSE(<str> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：反転した文字列を返します。
- 戻り型：string。
- 例：

```
> SELECT reverse('Spark SQL');
 LQS krapS
```

RLIKE

- 関数構文：

```
RLIKE(<str> string, <regexp> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：strがregexpに一致する場合はtrueを返し、それ以外の場合はfalseを返します。
- 戻り型：boolean
- 例：

```
> SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

FROM_CSV

- 関数構文：

```
FROM_CSV(<cvsStr> string, <schema> string, <options> map<string,
string>)
```

- サポートエンジン：SparkSQL
- 使用説明：指定されたcsvStrとschemaを持つ構造値を返します。
- 戻り型：struct
- 例：

```
> 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

- 関数構文：

```
SCHEMA_OF_CSV(<csvStr> string[, options map<string, string>])
```

- サポートエンジン：SparkSQL

- 使用説明：csv文字列のスキーマを返します。
- 戻り型：string。
- 例：

```
> SELECT schema_of_csv('1,abc');
 STRUCT<`_c0`: INT, `_c1`: STRING>
```

TO_CSV

- 関数構文：

```
TO_CSV(<expr> struct[, <options> map<string, string>])
```

- サポートエンジン：SparkSQL
- 使用説明：指定された構造値を持つcsv文字列を返します
- 戻り型：string。
- 例：

```
> 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

- 関数構文：

```
NGRAMS(<a> array<array<string>>, <N> integer, <K> integer, <pf>
integer)
```

- サポートエンジン：Presto。
- 使用説明：マーク化された文のセットから上位k個のN-gramを返します。
- 戻り型：array <struct <string,double>>。

CONTEXT_NGRAMS

- 関数構文：

```
CONTEXT_NGRAMS((array &lt;array &lt;string>>, array &lt;string>, int,
int))
```

- サポートエンジン：Presto。
- 使用説明：与えられたコンテキストN-gramに対して、トークン化された文のセットから上位k個のコンテキストN-gramを返します。
- 戻り型：array <struct <string,double>>。

集約関数

最終更新日：：2025-12-25 12:00:07

APPROX_COUNT_DISTINCT

- 関数構文：

```
APPROX_COUNT_DISTINCT(<expr> any [, <relativeSD>
integer|double|decimal])
```

- サポートエンジン：SparkSQL
- 使用説明：HyperLogLog++が推定した基数を返します。relativeSDは許容される最大相対標準偏差を定義します。
- 戻り値の型：bigint
- 例：

```
SELECT approx_count_distinct(col1) FROM (VALUES (1), (1), (2), (2),
(3)) tab(col1);
 3
```

AVG

- 関数構文：

```
AVG(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された平均値を返します。
- 戻り値の型：double
- 例：

```
> 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

- 関数構文：

```
CORR(<expr> integer|double|decimal, <expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：一連の数値ペア間のピアソン相関係数を返します。
- 戻り値の型：double
- 例：

```
> SELECT corr(c1, c2) FROM (VALUES (3, 2), (3, 3), (6, 4)) as tab(c1,
c2);
 0.8660254037844387
```

COUNT

- 関数構文：

```
COUNT(*)
COUNT([DISTINCT] <col1> ANY, <col2> ANY, ...)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：
  COUNT(*)：検索された行の総数を返します。nullを含む行も含まれます。
  COUNT(<col1> ANY, <col2> ANY, ...)：指定された式がすべて非nullである行数を返します。
  COUNT([DISTINCT] <col1> ANY,<col2> ANY, ...)：指定された式が一意かつ非nullである行数を返します。
- 戻り値の型：integer
- 例：

```
> 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

- 関数構文：

```
COUNT_IF(<expr> ANY)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：式がTRUEである行数を返します。
- 戻り値の型：int
- 例：

```
> 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

- 関数構文：

```
COVAR_POP(<expr1> integer|double|decimal, <expr2>
integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：一組の数値ペアの母共分散を返します。
- 戻り値の型：double
- 例：

```
> SELECT covar_pop(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS
tab(c1, c2);
 0.6666666666666666
```

COVER_SAMP

- 関数構文：

```
COVAR_SAMP(<expr1> integer|double|decimal, <expr2>
```

```
integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：一組の数値ペアの標本共分散を返します。
- 戻り値の型：double
- 例：

```
> SELECT covar_samp(c1, c2) FROM (VALUES (1,1), (2,2), (3,3)) AS
tab(c1, c2);
 1.0
```

FIRST_VALUE

- 関数構文：

```
FIRST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの行の最初の値を返します。isIgnoreNullがtrueの場合、null以外の値のみが返されます。
- 戻り値の型：T
- 例：

```
> 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

- 関数構文：

```
FIRST(<expr> T[, <isIgnoreNull> boolean])
```

- サポートエンジン：SparkSQL
- 使用説明：exprの行の最初の値を返します。isIgnoreNullがtrueの場合、null以外の値のみが返されます。

- 戻り値の型：T
- 例：

```
> 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

- 関数構文：

```
KURTOSIS(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL
- 使用説明：グループの値に基づいて計算された尖度値を返します。
- 戻り値の型：double
- 例：

```
> 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

- 関数構文：

```
LAST_VALUE(<expr> T[, <isIgnoreNull> boolean])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprの行の最後の値を返します。isIgnoreNullがtrueの場合、null以外の値のみが返されます。
- 戻り値の型：T
- 例：

```
> 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

- 関数構文：

```
LAST(<expr> T[, <isIgnoreNull> boolean])
```

- サポートエンジン：SparkSQL
- 使用説明：exprの行の最後の値を返します。isIgnoreNullがtrueの場合、null以外の値のみが返されます。
- 戻り値の型：T
- 例：

```
> 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

- 関数構文：

```
MEAN(<expr> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された平均値を返します。
- 戻り値の型：double
- 例：

```
> 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

- 関数構文：

```
PERCENTILE(<col> ANY, <percentage>
integer|double|decimal|array<double> [, <frequency> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：指定されたパーセンテージ下での数値列colの正確なパーセンタイル値を返します。percentage値は0.0から1.0の間でなければなりません。frequencyは正の整数である必要があります。
- 戻り値の型：double
- 例：

```
> 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

- 関数構文：

```
SKEWNESS(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL
- 使用説明：グループの値に基づいて計算された歪度を返します。
- 戻り値の型：double
- 例：

```
> 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

- 関数構文：

```
PERCENTILE_APPROX(<col> integer|double|decimal, <percentage>
double|array<double>[, <accuracy> integer])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：数値列colの近似パーセンタイルを返します。この値は、colの値を最小から最大に並べ替えた際に、指定したパーセンテージ以下の値となる最小値です。percentageの値は0.0から1.0の間でなければなりません。accuracyパラメータ（デフォルト値：10000）は正の数値リテラルで、メモリ使用量と引き換えに近似精度を制御します。精度値が高いほど近似精度が向上し、1.0/accuracyが近似相対誤差となります。percentageが配列の場合、配列内の各値は0.0から1.0の間でなければなりません。この場合、指定されたpercentage配列に対応する列colの近似パーセンタイル配列が返されます。
- 戻り値の型：integer |array<integer>
- 例：

```
> 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

- 関数構文：

```
APPROX_PERCENTILE(<col> integer|double|decimal, <percentage>
double|array<double>[, <accuracy> integer])
```

- サポートエンジン：SparkSQL、Presto

- 使用説明：数値列colの近似パーセンタイルを返します。この値は、colの値を最小から最大に並べ替えた際に、指定したパーセンテージ以下の値となる最小値です。percentageの値は0.0から1.0の間でなければなりません。accuracyパラメータ（デフォルト値：10000）は正の数値リテラルで、メモリ使用量と引き換えに近似精度を制御します。精度値が高いほど近似精度が向上し、1.0/accuracyが近似相対誤差となります。percentageが配列の場合、配列内の各値は0.0から1.0の間でなければなりません。この場合、指定されたpercentage配列に対応する列colの近似パーセンタイル配列が返されます。
- 戻り値の型：integer |array<integer>
- 例：

```
> SELECT APPROX_PERCENTILE(col, array(0.5, 0.4, 0.1), 100) FROM
(VALUES (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

- 関数構文：

```
MAX(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colの最大値を返します。
- 戻り値の型：colと一致します。
- 例：

```
> SELECT max(col) FROM (VALUES (10), (50), (20)) AS tab(col);
 50
```

MAX_BY

- 関数構文：

```
MAX_BY(<x> T, <y> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：yの最大値に関連付けられたx値を返します。

- 戻り値の型：T
- 例：

```
> SELECT max_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c',
20))) AS tab(x, y);
 b
```

MIN

- 関数構文：

```
MIN(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colの最小値を返します。
- 戻り型：colと一致
- 例：

```
> SELECT min(col) FROM (VALUES (10), (50), (20)) AS tab(col);
 10
```

MIN_BY

- 関数構文：

```
MIN_BY(<x> T, <y> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：yの最小値に関連付けられたx値を返します。
- 戻り値の型：T
- 例：

```
> SELECT min_by(x, y) FROM (VALUES (('a', 10)), (('b', 50)), (('c',
20))) AS tab(x, y);
 a
```

STD

- 関数構文：

```
STD(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto

使用説明

SparkSQL：グループの値に基づいて計算された標本標準偏差を返します。
Presto：グループの値に基づいて計算された母標準偏差を返します。

- 戻り値の型：double
- 例：

```
> SELECT std(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

STDDEV

- 関数構文：

```
STDDEV(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された標本標準偏差を返します。
- 戻り値の型：double
- 例：

```
> SELECT stddev(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

STDDEV_POP

- 関数構文：

```
STDDEV_POP(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された母集団標準偏差を返します。

- 戻り値の型：double
- 例：

```
> SELECT stddev_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 0.816496580927726
```

STDDEV_SAMP

- 関数構文：

```
STDDEV_SAMP(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された標本標準偏差を返します。
- 戻り値の型：double
- 例：

```
> SELECT stddev_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

SUM

- 関数構文：

```
SUM(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された合計値を返します。
- 戻り型：colと一致
- 例：

```
> 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

- 関数構文：

```
VARIANCE(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された標本分散を返します。
- 戻り値の型：double
- 例：

```
> SELECT VARIANCE(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

VAR_POP

- 関数構文：

```
VAR_POP(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された母分散を返します。
- 戻り値の型：double
- 例：

```
> SELECT var_pop(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 0.6666666666666666
```

VAR_SAMP

- 関数構文：

```
VAR_SAMP(<col> integer|double|decimal)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：グループの値に基づいて計算された標本分散を返します。
- 戻り値の型：double

- 例：

```
> SELECT var_samp(col) FROM (VALUES (1), (2), (3)) AS tab(col);
 1.0
```

HISTOGRAM_NUMERIC

- 関数構文：

```
HISTOGRAM_NUMERIC(<col> integer, <nb> integer)
```

- サポートエンジン：Presto
- 使用説明：nb個の不均等間隔のビンを使用して、グループ内の数値列のヒストグラムを計算します。出力は、ビンの中心と高さを表すサイズnbの（x、y）座標配列です。
- 戻り型：array<struct {'x','y'}>
- 例：

```
> 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

- 関数構文：

```
COLLECT_LIST(<col> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：非ユニーク要素のリストを収集して返します。
- 戻り値の型：array<T>
- 例：

```
> SELECT collect_list(col) FROM (VALUES (1), (2), (1)) AS tab(col);
 [1,2,1]
```

COLLECT_SET

- 関数構文：

```
COLLECT_SET(<col> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ユニークな要素のセットを収集して返します。
- 戻り値の型：array<T>
- 例：

```
> SELECT collect_set(col) FROM (VALUES (1), (2), (1)) AS tab(col);
 [1,2]
```

COUNT_MIN_SKETCH

- 関数構文：

```
COUNT_MIN_SKETCH(<col> T, <eps> double, <confidence> double, <seed>
integer)
```

- サポートエンジン：SparkSQL
- 使用説明：指定されたesp、confidence、seedを持つ列のcount min sketchを返します。結果はバイナリ形式で、使用前にCountMinSketchに逆シリアル化できます。
- 戻り値の型：binary
- 例：

```
> SELECT hex(count_min_sketch(col, 0.5d, 0.5d, 1)) FROM (VALUES (1),
(2), (1)) AS tab(col);
0000000100000000000000030000000100000004000000005D8D6AB900000000000000
0000000000000000200000000000000100000000000000000
```

EVERY

- 関数構文：

```
EVERY(<col> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colのすべての値がtrueの場合、trueを返します。
- 戻り値の型：boolean

- 例：

```
> 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

- 関数構文：

```
BOOL_AND(<col> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colのすべての値がtrueの場合、trueを返します。
- 戻り値の型：boolean
- 例：

```
> 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

- 関数構文：

```
<expr1> AND <expr2>
```

- サポートエンジン：SparkSQL、Presto

使用説明

論理積

- 戻り値の型：boolean
- 例：

```
> SELECT true and true;
 true
> SELECT true and false;
 false
> SELECT true and NULL;
 NULL
> SELECT false and NULL;
 false
```

OR

- 関数構文：

```
<expr1> OR <expr2>
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：論理和
- 戻り値の型：boolean
- 例：

```
> SELECT true or false;
 true
> SELECT false or false;
 false
> SELECT true or NULL;
 true
> SELECT false or NULL;
 NULL
```

ANY

- 関数構文：

```
ANY(<col> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colの少なくとも1つの値がtrueの場合、trueを返します。
- 戻り値の型：boolean
- 例：

```
> 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

- 関数構文：

```
SOME(<col> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colの少なくとも1つの値がtrueの場合、trueを返します。
- 戻り値の型：boolean
- 例：

```
> 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

- 関数構文：

```
BOOL_OR(<col> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：colの少なくとも1つの値がtrueの場合、trueを返します。

- 戻り値の型：boolean
- 例：

```
> 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

- 関数構文：

```
BIT_AND(<col> integer|bigint)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての非null入力値のビット単位ANDを返します。存在しない場合はnullを返します。
- 戻り型：colと一致
- 例：

```
> SELECT bit_and(col) FROM (VALUES (3), (5)) AS tab(col);
 1
```

BIT_OR

- 関数構文：

```
BIT_OR(<col> integer|bigint)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての非null入力値のビット単位ORを返します。存在しない場合はnullを返します。
- 戻り型：colと一致
- 例：

```
> SELECT bit_or(col) FROM (VALUES (3), (5)) AS tab(col);
 7
```

BIT_XOR

- 関数構文：

```
BIT_XOR(<col> integer|bigint)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての非null入力値のビット単位XORを返します。存在しない場合はnullを返します。
- 戻り型：colと一致
- 例：

```
> SELECT bit_xor(col) FROM (VALUES (3), (5)) AS tab(col);
 6
```

ARG_MIN

- 関数構文：

```
ARG_MIN(<col1>, <col2> | expr(col2))
```

- サポートエンジン：SparkSQL
- 使用説明：指定された列col1の最小値を持つ行を、指定された計算式に従って返します。
  - ソート可能な列名。定数の場合、任意の値が返されます。複数の最小値がある場合、ランダムに行が返されます。
- 戻り値の型：col2またはexpr(col2)の型と一致します。
- 例：

```
> 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

- 関数構文：

```
ARG_MAX(<col1>, <col2> | expr(col2))
```

- サポートエンジン：SparkSQL
- 使用説明：指定された列col1の最大値を持つ行を、指定された計算式に従って返します。
  - col1: ソート可能な列名。定数の場合、任意の値が返されます。最大値が複数ある場合、ランダムに行が返されます。
- 戻り値の型：col2またはexpr(col2)の型と一致します。
- 例：

```
> 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

- 関数構文：

```
MAP_UNION_SUM(map<k, v> input)
```

- サポートエンジン：SparkSQL
- 使用説明：指定した列で、mapをkey値に基づいてすべてのvalueを合計します。
  - 注意1：nullのデータを集計することが許可されており、すべてがnullの場合、出力はnullになります。
  - 注意2：keyのvalueがnullであることを許可し、その場合、数字0に変換されます。
- 戻り値の型：入力の型と一致します。
- 例：

```
> 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}
```

ウィンドウ関数

最終更新日：：2025-12-25 12:00:07

row_number

- 関数構文：

```
row_number()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：各行に一意の連続番号を割り当てます
- 戻り値の型：int
- 例：

```
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

- 関数構文：

```
rank()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ある値が一連の値の中で占める順位を計算します。同順位がある場合、順位シーケンスに空位が生じます。
- 戻り値の型：int
- 例：

```
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

- 関数構文：

```
  dense_rank()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ある値が一連の値の中で占める順位を計算します。同順位がある場合、関数rankとは異なり、dense_rankは順位シーケンスに空位を生じさせません。
- 戻り値の型：int
- 例：

```
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()

- 関数構文：

```
  percent_rank()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ある値が一連の値の中で占めるパーセントランクを計算します。戻り値は0から1の間の小数で表されます。
- 戻り値の型：double
- 例：

```
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

- 関数構文：

```
cume_dist()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：特定の値がパーティション内のすべての値に対して相対的に占める位置を計算します。
- 戻り値の型：double
- 例：

```
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

- 関数構文：

```
  first_value(col)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：パーティション内の列の最初のデータの値を返します
- 戻り値の型：col列のデータ型
- 例：

```
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

- 関数構文：

```
  last_value(col)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：パーティション内の列の最後のデータの値を返します
- 戻り値の型：int
- 例：

```
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

- 関数構文：

```
 lag(col[, n [, default]])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ウィンドウ内の現在の行から上にn行目の値を返します。nのデフォルト値は1、defaultのデフォルト値はnullです。n行目の値がnullの場合、nullを返します。そのようなオフセット行が存在しない場合（例えば、オフセットが1でウィンドウの最初の行に上に行がない場合）、defaultを返します。最初の引数は列名、2番目の引数は前のn行、3番目の引数はデフォルト値です。
- 戻り値の型：col列のデータ型
- 例：

```
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

- 関数構文：

```
lead(col[, n[, default]])
```

- サポートエンジン：SparkSQL、Presto
- 使用方法：ウィンドウ内の現在の行から下にn行目の値を返します。nのデフォルト値は1、defaultのデフォルト値はnullです。n行目の値がnullの場合、nullを返します。そのようなオフセット行が存在しない場合（例えば、オフセットが1でウィンドウの最後の行に下の行がない場合）、defaultを返します。最初の引数は列名、2番目の引数は前のn行、3番目の引数はデフォルト値です。
- 戻り値の型：col列のデータ型
- 例：

```
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

- 関数構文：

```
nth_value(col[, n])
```

- サポートエンジン：SparkSQL、Presto
- 使用方法：ウィンドウの先頭からn行目の値を返します。nは1から始まります。ignoreNulls=trueの場合、n行目を検索する際にnullをスキップします。それ以外の場合、各行がnとしてカウントされます。そのようなn行目が存在しない場合（例えば、nが10でウィンドウサイズが10未満の場合）、nullを返します。最初の引数は列名、2番目の引数は前のn行目です。
- 戻り値の型：col列のデータ型
- 例：

```
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

- 関数構文：

```
  ntile(n)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：ウィンドウパーティションの行をn個のバケットに分割し、行が属するバケット番号を1からnの範囲で返します
- 戻り値の型：int
- 例：

```
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

- 関数構文：

```
  CLUSTER_SAMPLE(<int> N[, <int> M]) over (PARTITION BY col1 ORDER by
col2)
```

- サポートエンジン：SparkSQL
- 使用説明：ウィンドウ内で指定された比率または数量でサンプリングします。
  - N：必須、int 型、N のみ指定の場合、N 件のデータをサンプリングすることを示します。サンプリング結果は N 件に近くなりますが、必ずしも N 件になるとは限りません。

- M：任意、int 型、M が指定されている場合、M/N * ウィンドウ内の総データ数のデータをサンプリングすることを示します。サンプリング結果は M/N * 総データ数に近くなります。

- 戻り値の型：boolean 型、true はサンプリングされたことを示し、false はサンプリングされなかったことを示します。
- 例：

```
> 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
```

他の関数

最終更新日：：2025-12-25 12:00:07

FIELD

- 関数構文：

```
FIELD(<val> T, <val1> T, <val2> T, ...)
```

- サポートエンジン：Presto
- 使用説明：val1、val2...リスト内のvalのインデックスを返します。見つからない場合は0を返します。
- サポート対象はすべてのプリミティブ型で、str.equals（x）を使用して引数を比較します。valがNULLの場合、戻り値は0です。
- 戻り値の型：integer
- 例：

```
select field('world', 'say', 'hello', 'world');
 3
```

COALESCE

- 関数構文：

```

COALESCE(<expr1> T, <expr2> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：存在する場合、最初の空でないパラメータを返します。それ以外の場合はnullを返します。
- 戻り値の型：integer
- 例：

```
> SELECT coalesce(NULL, 1, NULL);
 1
```

EXPLODE

- 関数構文：

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

- サポートエンジン：SparkSQL
- 使用説明：array型のexprの要素を複数の行に分割するか、またはmap型のexprを複数の行と列に分割します。配列の要素にはデフォルトの列名colを使用し、マップの要素にはkeyとvalueを使用します。
- 戻り値の型：row(col T) | row(key K, value V)
- 例：

```
SELECT explode(array(10, 20));
 10
 20
```

EXPLODE_OUTER

- 関数構文：

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

- サポートエンジン：SparkSQL
- 使用説明：array型のexprの要素を複数の行に分割するか、またはmap型のexprを複数の行と列に分割します。配列の要素にはデフォルトの列名colを使用し、マップの要素にはkeyとvalueを使用します。
- 戻り値の型：row(col T) | row(key K, value V)
- 例：

```
SELECT explode_outer(array(10, 20));
 10
 20
```

GREATEST

- 関数構文：

```
GREATEST(<expr1> T, <expr2> T, ...>)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての引数の中で最大値を返し、空値はスキップします。
- 戻り値の型：T
- 例：

```
> SELECT greatest(10, 9, 2, 4, 3);
 10
```

IF

- 関数構文：

```
IF(<expr1> boolean, <expr2> T, <expr3> U)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1の評価結果がtrueの場合、expr2を返し、それ以外の場合はexpr3を返します。
- 戻り値の型：T|U
- 例：

```
> SELECT if(1 < 2, 'a', 'b');
 a
```

INLINE

- 関数構文：

```
INLINE(a array<struct<f1:T1,...,fn:Tn>>)
```

- サポートエンジン：SparkSQL
- 使用説明：構造体配列をテーブルに分解します。デフォルトでは列名col1、col2などが使用されます。
- 戻り値の型：row(T1, ..., Tn)
- 例：

```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));
 1  a
```

```
 2  b
```

INLINE_OUTER

- 関数構文：

```
INLINE_OUTER(a array<struct<f1:T1,...,fn:Tn>>)
```

- サポートエンジン：SparkSQL
- 使用説明：構造体配列をテーブルに分解します。デフォルトでは列名col1、col2などが使用されます。
- 戻り値の型：row(T1, ..., Tn)
- 例：

```
> SELECT inline(array(struct(1, 'a'), struct(2, 'b')));
 1  a
 2  b
```

IN

- 関数構文：

```
<expr1> IN(<expr2> T, <expr3> T, ...)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1が任意のexprnと等しい場合、trueを返します。
- 戻り値の型：boolean
- 例：

```
> SELECT 1 in(1, 2, 3);
 true
> SELECT 1 in(2, 3, 4);
 false
```

ISNAN

- 関数構文：

```
ISNAN(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprがNaNの場合、trueを返し、それ以外の場合はfalseを返します。
- 戻り値の型：boolean
- 例：

```
> SELECT isnan(cast('NaN' as double));
 true
```

IFNULL

- 関数構文：

```
IFNULL(<expr1> T, <expr2> U)
```

- サポートエンジン：SparkSQL
- 使用説明：expr1がnullの場合、expr2を返し、それ以外の場合はexpr1を返します。
- 戻り値の型：T|U
- 例：

```
> SELECT ifnull(NULL, array('2'));
 ["2"]
```

ISNULL

- 関数構文：

```
ISNULL(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprがnullの場合、trueを返し、それ以外の場合はfalseを返します。
- 戻り値の型：boolean
- 例：

```
> SELECT isnull(1);
 false
```

ISNOTNULL

- 関数構文：

```
ISNOTNULL(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprがnullでない場合、trueを返し、それ以外の場合はfalseを返します。
- 戻り値の型：boolean
- 例：

```
> SELECT isnotnull(1);
 true
```

LEAST

- 関数構文：

```
LEAST(<expr1> T, <expr2> T, ...)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：すべての引数の中で最小値を返し、nullはスキップします。
- 戻り値の型：T
- 例：

```
> SELECT least(10, 9, 2, 4, 3);
 2
```

NANVL

- 関数構文：

```
NANVL(<expr1> T, <expr2> U)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：nanvl（expr1、expr2）、expr1がNaNでない場合はexpr1を返し、それ以外の場合はexpr2を返します。
- 戻り値の型：T|U
- 例：

```
> SELECT nanvl(cast('NaN' as double), 123);
 123.0
```

NULLIF

- 関数構文：

```
NULLIF(<expr1> T, <expr2> U)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1がexpr2と等しい場合、nullを返し、それ以外の場合はexpr1を返します。
- 戻り値の型：T
- 例：

```
> SELECT nullif(2, 2);
 NULL
```

NVL

- 関数構文：

```
NVL(<expr1> T, <expr2> U)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1がnullの場合、expr2を返し、それ以外の場合はexpr1を返します。
- 戻り値の型：T|U

- 例：

```
> SELECT nvl(NULL, array('2'));
 ["2"]
```

NVL2

- 関数構文：

```
NVL2(<expr1> T1, <expr2> T2, <expr3> T3)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr1がnullでない場合、expr2を返し、それ以外の場合はexpr3を返します。
- 戻り値の型：T2|T3
- 例：

```
> SELECT nvl2(NULL, 2, 1);
 1
```

POSEXPLODE

- 関数構文：

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

- サポートエンジン：SparkSQL
- 使用説明：array型のexprの要素を複数の行に分割するか、またはmap型のexprを複数の行と列に分割します。位置を表す列名posを使用し、配列の要素にはデフォルトの列名colを使用し、マップの要素にはkeyとvalueを使用します。
- 戻り値の型：row(pos integer, col T)|row(row integer, key K, value V)
- 例：

```
> SELECT posexplode(array(10,20));
 0  10
 1  20
```

POSEXPLODE_OUTER

- 関数構文：

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

- サポートエンジン：SparkSQL
- 使用説明：array型のexprの要素を複数の行に分割するか、またはmap型のexprを複数の行と列に分割します。位置を表す列名posを使用し、配列の要素にはデフォルトの列名colを使用し、マップの要素にはkeyとvalueを使用します。
- 戻り値の型：row(pos integer, col T)|row(row integer, key K, value V)
- 例：

```
> SELECT posexplode_outer(array(10,20));
 0  10
 1  20
```

STACK

- 関数構文：

```
STACK(<n> integer, <expr0> T0, ..., <expr1> T1)
```

- サポートエンジン：SparkSQL
- 使用説明：stack(n, expr1, ..., exprk) – expr1, ..., exprkをn行に分割します。デフォルトでは列名col0、col1などが使用されます。
- 戻り値の型：row(col0 T0, ..., coln Tn)
- 例：

```
> SELECT stack(2, 1, 2, 3);
 1  2
 3  NULL
```

ASSERT_TRUE

- 関数構文：

```
ASSERT_TRUE(<expr> boolean)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprがtrueでない場合、例外をスローします。
- 戻り値の型：boolean
- 例：

```
> SELECT assert_true(0 < 1);
 NULL
```

RAISE_ERROR

- 関数構文：

```
RAISE_ERROR(<error> string)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：expr例外をスローします。
- 戻り値の型：string
- 例：

```
> SELECT raise_error('custom error message');
 java.lang.RuntimeException
 custom error message
```

SPARK_PARTITION_ID

- 関数構文：

```
SPARK_PARTITION_ID()
```

- サポートエンジン：SparkSQL
- 使用説明：現在のパーティションidを返します。
- 戻り値の型：integer

- 例：

```
> SELECT spark_partition_id();
 0
```

INPUT_FILE_NAME

- 関数構文：

```
INPUT_FILE_NAME()
```

- サポートエンジン：SparkSQL
- 使用説明：読み取り中のファイル名を返します。利用できない場合は空の文字列を返します。
- 戻り値の型：string
- 例：

```
> SELECT input_file_name();
```

INPUT_FILE_BLOCK_START

- 関数構文：

```
INPUT_FILE_BLOCK_START()
```

- サポートエンジン：SparkSQL
- 使用説明：読み取り中のブロックの開始オフセットを返します。利用できない場合は-1を返します。
- 戻り値の型：integer
- 例：

```
> SELECT input_file_block_start();
 -1
```

INPUT_FILE_BLOCK_LENGTH

- 関数構文：

```
INPUT_FILE_BLOCK_LENGTH()
```

- サポートエンジン：SparkSQL
- 使用説明：読み取り中のブロックの長さを返します。利用できない場合は-1を返します。
- 戻り値の型：integer
- 例：

```
> SELECT input_file_block_length();
 -1
```

MONOTONICALLY_INCREASING_ID

- 関数構文：

```
MONOTONICALLY_INCREASING_ID()
```

- サポートエンジン：SparkSQL
- 使用説明：単調増加する64ビット整数を返します。生成されたIDは単調増加かつ一意であることが保証されますが、連続的ではありません。現在の実装では、パーティションIDを上位31ビットに配置し、下位33ビットで各パーティション内のレコード番号を表します。データフレームのパーティションが10億未満で、各パーティションのレコードが80億未満であると仮定しています。この関数は、その結果がパーティションIDに依存するため、非決定的です。
- 戻り値の型：bigint
- 例：

```
> SELECT monotonically_increasing_id();
 0
```

CURRENT_DATABASE

- 関数構文：

```
CURRENT_DATABASE()
```

- サポートエンジン：SparkSQL
- 使用説明：現在のデータベースを返します。
- 戻り値の型：string
- 例：

```
> SELECT current_database();
 default
```

CURRENT_CATALOG

- 関数構文：

```
CURRENT_CATALOG()
```

- サポートエンジン：SparkSQL
- 使用説明：現在のcatalogを返します
- 戻り値の型：string
- 例：

```
> SELECT current_catalog();
 spark_catalog
```

CURRENT_USER

- 関数構文：

```
CURRENT_USER()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：現在のユーザーを返します。
- 戻り値の型：string
- 例：

```
> SELECT current_user();
```

REFLECT

- 関数構文：

```
REFLECT(<class> string, <method> string[, <arg1> T1[, <arg2> T2,
...]])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：リフレクションを持つメソッドを呼び出します。
- 戻り値の型：string
- 例：

```
> select reflect('java.lang.Math', 'abs', -1);
 1
```

JAVA_METHOD

- 関数構文：

```
JAVA_METHOD(<class> string, <method> string[, <arg1> T1[, <arg2> T2,
...]])
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：リフレクションを持つメソッドを呼び出します。
- 戻り値の型：string
- 例：

```
> select JAVA_METHOD('java.lang.Math', 'abs', -1);
 1
```

VERSION

- 関数構文：

```
```

```
VERSION()
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：エンジンのバージョンを返します。
- 戻り値の型：string
- 例：

```
> select VERSION()
 3.0.0 rce61711a5fa54ab34fc74d86d521ecaeea6b072a
```

TYPEOF

- 関数構文：

```
TYPEOF(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprのデータ型を返します
- 戻り値の型：string
- 例：

```
> SELECT typeof(1);
 int
> SELECT typeof(array(1));
 array<int>
```

CAST

- 関数構文：

```
CAST(<expr> AS <type>)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprをtypeタイプに変換します
- 戻り値の型：<type>

- 例：

```
> SELECT cast('10' as int);
 10
```

BOOLEAN

- 関数構文：

```
BOOLEAN(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：exprをbooleanタイプに変換します
- 戻り値の型：boolean
- 例：

```
> SELECT boolean(1);
 true
```

BIGINT

- 関数構文：

```
BIGINT(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：強制型変換をbigintに
- 戻り値の型：bigint
- 例：

```
> select bigint(0);
 0
```

BINARY

- 関数構文：

```
BINARY(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：BINARYへの強制型変換
- 戻り値の型：binary
- 例：

```
> select binary(65);
 A
```

DOUBLE

- 関数構文：

```
DOUBLE(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：doubleへの強制型変換
- 戻り値の型：double
- 例：

```
select double(1);
 1.0
```

FLOAT

- 関数構文：

```
FLOAT(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：floatへの強制型変換
- 戻り値の型：float
- 例：

```
> select float(1);
 1.0
```

INT

- 関数構文：

```
INT(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：integer型への強制型変換
- 戻り値の型：integer
- 例：

```
> select int(1.0);
 1
```

SMALLINT

- 関数構文：

```
SMALLINT(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：強制型変換をsmallintに
- 戻り値の型：smallint
- 例：

```
select typeof(smallint(1));
 smallint
```

STRING

- 関数構文：

```
STRING(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：stringへの強制型変換
- 戻り値の型：string
- 例：

```
> select typeof(string(1));
 string
```

TINYINT

- 関数構文：

```
TINYINT(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：強制型変換をtinyintに
- 戻り値の型：tinyint
- 例：

```
> select typeof(tinyint(1));
 tinyint
```

DECIMAL

- 関数構文：

```
DECIMAL(<expr> T)
```

- サポートエンジン：SparkSQL、Presto
- 使用説明：decimalへの強制型変換
- 戻り値の型：decimal
- 例：

```
> select typeof(decimal(1));
 decimal(10, 0)
```

GET_IDCARD_BIRTHDAY

- 関数構文：

```
GET_IDCARD_BIRTHDAY(<string> idcardno)
```

- サポートエンジン：SparkSQL
- 使用説明：身分証明書番号から生年月日を取得する
  - idcardno：必須、string タイプ、15桁または18桁の身分証明書番号でなければなりません。身分証明書の合理性を検証します。null以外の他の型の入力は許可されません。入力がnullの場合、nullを返します。
- 戻り値の型：date
- 例（以下のテスト用身分証明書番号はランダムなサンプルであり、正しい身分証明書番号を表すものではありません）

```
> SELECT get_idcard_birthday('421081199001011222');
1990-01-01
```

GET_IDCARD_SEX

- 関数構文：

```
GET_IDCARD_SEX(<string> idcardno)
```

- サポートエンジン：SparkSQL
- 使用説明：身分証明書番号から性別を取得する
  - idcardno：必須、string タイプ、15桁または18桁の身分証明書番号でなければなりません。身分証明書の合理性を検証します。null以外の他の型の入力は許可されません。入力がnullの場合、nullを返します。
- 戻り値の型：string
- 例：（以下のテスト用身分証明書番号はランダムなサンプルであり、正しい身分証明書番号を表すものではありません）

```
> SELECT get_idcard_birthday('421081199001011222');
```

GET_IDCARD_AGE

- 関数構文：

```
GET_IDCARD_AGE(<string> idcardno)
```

- サポートエンジン：SparkSQL
- 使用説明：身分証明書番号から年齢を取得する
  - idcardno：必須、string タイプ、15桁または18桁の身分証明書番号でなければなりません。身分証明書の合理性を検証します。null以外の他の型の入力は許可されません。入力がnullの場合、nullを返します。
- 戻り値の型：int
- 例：（以下のテスト用身分証明書番号はランダムなサンプルであり、正しい身分証明書番号を表すものではありません）

```
> SELECT get_idcard_age('421081197001021233');
53
```

MAX_PT

- 関数構文：

```
MAX_PT(<const string> tableFullName)
```

- サポートエンジン：SparkSQL
- 使用説明：指定されたパーティションテーブルの最大値を取得します
  - tableFullName：必須、string タイプ、定数値である必要があります。そうでない場合はエラーが発生します。tableFullName は catalog.db.table の3つの部分で構成され、catalog と db を省略すると現在のセッションの設定がデフォルトで使用されます。完全な記述を推奨します。
  - 入力されたテーブルがパーティションテーブルでない場合、エラーが発生します。
  - この関数は**データがある**最大のパーティションのみを返します。データの有無はメタデータセンターに関わるため、**外部テーブルの場合、ANALYZE ステートメントを実行してパーティションの統計情報をメタデータに報告したかどうかに注意する必要があります。**
  - パーティション値は辞書順に並べ替えて最大値を取得します。複数レベルのパーティションがある場合、**第一レベルのパーティション**のみをソート対象とします。
- 戻り値の型：string
- 例：

```
> SELECT max_pt('test.tableName');
20231024
> select * from test.tableName where dt=max_pt('test.tableName');
select * from test.tableName where dt='20231014'; と同等です。
```

TRANS_ARRAY

- 関数構文：

```
TRANS_ARRAY(<int> numKeys, <string> separator, <key1>, <key2>, ...,
<col1>, <col2>, ...)
```

- サポートエンジン：SparkSQL
- 使用説明：指定された複数の列colsを分割して転置し、複数の行に変換します。また、転置のキーとして一部の列keysを指定することもサポートしています。
  - numKeys：int 型、必須、転置用のキー列の数を示し、0以上でなければなりません。
  - separator：string タイプ、必須項目。colsが文字列の場合、separatorに基づいて分割する必要があります。colsが配列の場合、このパラメータは任意の値を入力できます。
  - keys：任意のタイプ、数はnumKeysによって決定されます。
  - cols：文字列型または配列型で、指定された列からkeysを除いたすべてがcolsと見なされます。つまり、分割および転置される列です。**すべてのcolsの型は同じでなければなりません**。つまり、すべてが文字列、またはすべてが配列、あるいは特殊なケースとして文字列と文字列配列の2種類の組み合わせもサポートされます。colsがない場合、1行のみ出力されます。
  - 注意1：keysとcolsの数の合計は0より大きくなければなりません。
  - 注意2：colsを分割した後の長さが等しくない場合、最終的に転置後の行数は最も長い列に合わせ、他の列はNULLで補完されます。
- 戻り値の型：任意の型。keysの型は変更されず、colsの型は文字列または配列の要素型です。
- 例：

```
> 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

- 関数構文：

```
TRANS_COLS(<int> numKeys, <key1>, <key2>, ..., <col1>, <col2>, ...)
```

- サポートエンジン：SparkSQL
- 使用説明：指定された複数の列colsを転置し、複数の行に変換します。また、転置のキーとして一部の列keysを指定することもサポートしています。
  - numKeys：int型、必須、転置用のkeys列の数を示し、0以上でなければなりません。
  - keys：任意のタイプ、数はnumKeysによって決定されます。
  - cols：任意のタイプ、指定された列からkeysを除いたすべてがcolsと見なされます、つまり転置する列です。**すべてのcolsのタイプは同じでなければなりません**。colsがない場合、1行のみ出力されます。
  - 注意：keysとcolsの数の合計は0より大きくなければなりません。
- 戻り値の型：任意の型。出力には、転置されたすべての行における行番号を示すidx列が含まれ、keysの型は変更されず、colsの型も変更されません。
- 例：

```
> 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

- 関数構文：

```
SAMPLE(<int/long> totalParts[, <int/long> pointParts][, <col1>,
<col2>, ...])
```

- サポートエンジン：SparkSQL

- 使用方法：サンプリング関数を使用して、グローバルまたは指定された列のハッシュ値をtotalPartsに分割し、指定された第pointPartsの結果を返します。
  - totalParts：int/long型、必須、分割する必要がある総パーティション数を示します。
  - pointParts：int/long型、任意、返すデータの何番目の部分を選択するかを示し、デフォルト値は1で、totalParts以下である必要があります。
  - cols：任意のタイプ、任意指定、データを分割する際のキーとして任意の列を指定可能。これらの列のハッシュ値に基づいて分割されます。
  - 注1：colsが指定されていない場合、または現在指定されているcolsがすべてNULLの場合、データの偏りを防ぐために、**ハッシュ値の代わりにランダム値を使用**して計算します。一方、colsが指定されている場合、1列でもNULLでない列があると、ハッシュ値の計算およびパーティション結果に影響を与えます。
  - 注意2：ハッシュ値とランダム値のシードが固定されているため、データの順序が一致すれば、繰り返し実行しても単一パーティション内のサンプリングデータは同じになります。小ファイルの結合が行われるシナリオでは、サンプリング順序が変化する可能性があるため、この場合は列を指定してサンプリングすることをお勧めします。
- 戻り値の型：boolean、trueはサンプリング済み、falseはサンプリング未実施を表します。
- 例：

```
> 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

- 関数構文：

```
TO_CHAR(<boolean|int|long|double|decimal> data [, <string> format])
TO_CHAR(<date|timestamp> data, string format)
```

- サポートエンジン：SparkSQL
- 使用説明1：指定されたテンプレートに従って数値をフォーマットします。formatは省略可能で、その場合は数値を直接文字列に変換します。それ以外の場合、formatの定義は以下の通りです：
  - '0' または '9': 数字のプレースホルダー。文字列の先頭の場合、0がプレースホルダーのとき数字がない場合は0で置き換え、9がプレースホルダーのとき数字がない場合はスペースで置き換える。文字列の末尾の場合、数字がない場合は一律0で置き換える。

- '.' または 'D': 小数点のプレースホルダーで、1回のみ出現できます。
- 「、」または「G」: カンマ、例えば百、百万のプレースホルダーで、左右は必ず0-9の数字でなければなりません。
- '$': ドル記号、1回のみ出現可能。
- 'S' または 'MI': 負号と正号のプレースホルダーで、文字列の先頭または末尾に最大1回のみ出現できます。
- 'PR': 文字列の末尾に一度だけ出現することが許可され、入力データが負の符号の場合、数字を正に変換して山括弧を追加します。

- 使用説明2: 指定されたテンプレートに従って日付または時刻をフォーマットします。formatは省略不可で、定義は以下の通りです:
  - yyyyMMdd HH:mm:ss.SSSの任意の文字で構成されるテンプレート。
  - 詳細はこちら: https://spark.apache.org/docs/latest/sql-ref-datetime-pattern.html
- 使用説明3: TO_CHARは最初の引数の型を暗黙的に変換することをサポートします。最初の引数がstring型の場合、関数はデフォルトでtimestampに変換され、テンプレートに従って日付または時刻をフォーマットするロジックが実行されます。
- 戻り値の型: string型
- 例:

```
> 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 組み込み関数

最終更新日：：2025-12-25 12:00:07

データレイクコンピューティング DLC は、統一関数 に加えて、Presto の組み込み関数もサポートしています。

Presto 組み込み関数アプリケーションを有効にする方法

方法1：データ探索でデータエンジンに関数設定を行います

1. データレイクコンピューティング DLC コンソールにログインし、サービス地域を選択します。
2. データ探索に入り、データエンジンを選択します。エンジンカーネルがPrestoの場合、高度な設定でパラメータUSEHIVEFUNCTIONを選択し、このパラメータをfalseに設定すると、このデータエンジンを使用してSQLタスクを実行する際にPrestoの組み込み関数を使用できます。

> **注意**
> 現在のクエリセッションでは、このデータエンジンを使用するすべてのクエリタスクでPrestoの組み込み関数を使用できます。

方法二：SQL文にパラメータを追加する

特定のSQLタスクでPrestoの組み込み関数を呼び出したい場合、SQLタスクに設定情報を追加することで実現できます。
例：

```
SELECT/*+OPTIONS('useHiveFunction'='false')*/prestofunc(xx)
```

方法三：API使用時に設定パラメータを追加する

task構造体のconfigにkv、useHiveFunction=falseを設定します。

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 KVPair	否	任务的配置信息

例：

```
Stringstatement="SELECTdate_add('week',3,TIMESTAMP'2001-08-
2203:04:05.321')";
TasksInfotask=newTasksInfo();
task.setTaskType("SQLTask");
task.setSQL(Base64.getEncoder().encodeToString(statement.getBytes()));
//以下のパラメータ設定を追加
KVPairpair=newKVPair();
pair.setKey("useHiveFunction");
pair.setValue("false");
task.setConfig(newKVPair[]{pair});

CreateTasksRequestrequest=newCreateTasksRequest();
request.setDatabaseName("");
request.setDataEngineName(PRESTO_ENGINE);
request.setTasks(task);
```

APIドキュメントを参照： タスク作成 。

方法四：JDBCを使用してタスク作成時にパラメータを追加する

JDBCURL パスに `&presto.USEHIVEFUNCTION=true` パラメータまたは `info.setProperty("presto.USEHIVEFUNCTION","false");` を追加します。

```
Connectionconnection=
DriverManager.getConnection("jdbc:dlc:dlc.tencentcloudapi.com?
task_type=SQLTask&x省略中间参数xx&presto.USEHIVEFUNCTION=true",info);
```

または

```
info.setProperty("presto.USEHIVEFUNCTION","false");
info.setPropertv("user","");
info.setProperty("password","");
```

サポートされているPresto組み込み関数のリスト

数学関数

関数名	戻り値	関数機能説明
abs(x)	x と一致	x の絶対値を返します
cbrt(x)	double	x の立方根を返します
ceil(x) | ceiling(x)	x と一致	x の最も近い整数を返します。例えば：ceil(2.2); –->3
cosine_similarity(x, y)	double	ベクトル x と y の類似度を返します 例えば：SELECT cosine_similarity(MAP(ARRAY['a'], ARRAY[1.0]), MAP(ARRAY['a'], ARRAY[2.0])); –->1.0
degrees(x)	double	ラジアンを角度に変換します
e()	double	定数 e を返します
exp(x)	double	x の e 乗を返します
floor(x)	x と一致	x の最も近い整数を返します。例えば：floor(2.2); –->2
ln(x)	double	x の自然対数を返します
log2(x)	double	x の2を底とする対数を返します
log10(x)	double	x の10を底とする対数を返します
mod(n, m)	n と一致	n を m で割った余りを返します
pi()	double	定数Piを返します
pow(x, p) | power(x, p)	double	x の p 乗を返します
radians(x)	double	角度xをラジアンに変換します
rand() | random()	double	0.0 <= x < 1.0 の範囲内のランダムな値を返します。
random(n)	n と一致	0 から n (nを含まない) までの間のランダムな数を返します

secure_rand() | secure_random()	double	0.0 <= x < 1.0 の範囲内の暗号化されたランダムな値を返します。
secure_random(lower, upper)	lower と一致	lower <= x < upper の範囲内の暗号化されたランダムな値を返します（ただし、lower < upper）
round(x)	x と一致	x を最も近い整数に丸めて返します
round(x, d)	x と一致	x を小数点以下 d 桁に丸めて返します
sign(x)	x と一致	x に対応する符号を返します。x > 0 の場合は 1、x < 0 の場合は –1、x = 0 の場合は 0 を返します。
sqrt(x)	double	x の平方根を返します
truncate(x)	bigint	小数点以下を切り捨てた数を返します 例えば：truncate(4.9) –->4
truncate(x, n)	double	x の小数点以下n桁を切り捨てて返します。n<0の場合は、小数点の左側から切り捨てます。 例えば：truncate(12.333, –1) –->10.0；truncate(12.333,0) –->12.0；truncate(12.333,1) –->12.3

バイナリ関数

関数名	戻り値	関数機能説明
length(b)	bigint	バイト単位で、bのバイナリ長を返します
concat(b1, ..., bN)	varbinary	b1,...,bNを連結したバイナリ値を返します
substr(b, start)	varbinary	b の start 位からバイナリ値を切り取り、start > 0 の場合は先頭から、start < 0 の場合は末尾から切り取ります
substr(b, start, length)	varbinary	b の start 位から length の長さのバイナリ値を切り取り、start > 0 の場合は先頭から、start < 0 の場合は末尾から切り取ります
to_base64(b)	varchar	バイナリデータ b を base64 文字列に変換します
from_base64(string)	varbinary	base64エンコードされた文字列をバイナリデータに変換します
to_base64url(b)	varchar	URLセーフな文字を使用して、バイナリデータ b を base64 文字列に変換します
from_base64url(strin	varbinary	URLセーフな文字を使用して、base64エンコードされた文

g)		字列をバイナリデータに変換します
from_base32(string)	varbinary	base32でエンコードされた文字列をバイナリデータに変換する
to_base32(b)	varchar	バイナリデータ b を base32 文字列に変換する
to_hex(b)	varchar	バイナリデータ b を16進数文字列に変換する
from_hex(string)	varbinary	16進数エンコードされた文字列をバイナリデータに変換する
lpad(b, size, padb)	varbinary	b の左側に padb を連結し、長さが size に達するまで繰り返したバイナリ数を返す
rpad(b, size, padb)	varbinary	b の右側に padb を連結し、長さが size に達するまで繰り返したバイナリ数を返す
crc32(b)	bigint	CRC32アルゴリズムを使用して式の巡回冗長検査値を計算する
md5(b)	varbinary	バイナリbのmd5値を計算する
sha1(b)	varbinary	バイナリbのsha1値を計算する
sha256(b)	varbinary	バイナリbのsha256値を計算する
sha512(b)	varbinary	バイナリbのsha512値を計算する
xxhash64(b)	varbinary	バイナリbのxxhash64値を計算する

ビット演算関数

関数名	戻り値	関数機能説明
bit_count(x, bits)	bigint	x における bit のビット数を返します
bitwise_and(x, y)	bigint	x と y の AND を返します
bitwise_not(x)	bigint	x のビット単位の NOT を返します
bitwise_or(x, y)	bigint	x と y の OR を返します
bitwise_xor(x, y)	bigint	x と y の XOR を返します
bitwise_left_shift(value, shift)	入力と一致	value の左シフト shift ビットの値を返します bitwise_left_shift(TINYINT '7', 2) --> 28
bitwise_right_shift(val	入力と一致	value の右シフト shift ビットの値を返します

ue, shift)		bitwise_right_shift(TINYINT '7', 2) -->1

文字列関数

関数名	戻り値	関数機能説明
chr(n)	varchar	n の Unicode 文字を返す
codepoint(string)	integer	文字列のUnicode値を返す 例えば：codepoint('0') -->48
concat(string1, ..., stringN)	varchar	string1,....,stringNの連結文字列を返す
length(string)	bigint	文字列の長さを返す
hamming_distance(string1, string2)	bigint	2つの（同じ長さの）文字列の対応する位置にある異なる文字の数を返します 例えば：hamming_distance(abc, art) -->2
levenshtein_distance(string1, string2)	bigint	2つの文字列間で、一方から他方に変換するために必要な最小の編集（挿入、削除、置換）回数を返します 例えば：levenshtein_distance('ab', 'abcde') -->3
lower(string)	varchar	stringを小文字に変換した文字列を返す
upper(string)	varchar	stringを大文字に変換した文字列を返す
ltrim(string)	varchar	文字列の左側のスペースを削除
rtrim(string)	varchar	文字列の後ろのスペースを削除
trim(string)	varchar	文字列からスペースを削除して返す
replace(string, search)	varchar	文字列からすべてのsearch文字を削除する
replace(string, search, replace)	varchar	search文字列をreplace文字列に置き換える replace('abcavc', 'a', 'c') --> 'cbccvc'
reverse(string)	varchar	文字列を逆順に並べ替えて返す 例えば：reverse('abc') --> 'cba'
lpad(string, size, padstring)	varchar	stringの左側にpadstringを連結し、文字列の長さがsizeに達するまで繰り返します。sizeがstringより小さい場合、stringを長さsizeの文字列に切り詰めます。
rpad(string, size, padstring)	varchar	stringの右側にpadstringを連結し、文字列の長さがsizeに達するまで繰り返します。sizeがstringより小さい場合、

		stringを長さsizeの文字列に切り詰めます。
split(string, delimiter)	array	delimiterで分割された文字列の配列を返します
split(string, delimiter, limit)	array	delimiterで分割された文字列をlimitのサイズ制限に従って配列にし、limit > 0の場合、配列の最後の要素には文字列の残りの内容がすべて含まれます 例えば：select split('ab-cd-efg', '-', 2) -->['ab','cd-efg']
split_part(string, delimiter, index)	varchar	delimiter で分割された文字列の配列において、index 位置の文字列を返します。フィールドインデックスは1から始まります。index > 配列の長さの場合、null を返します。 例えば：select split_part('ab-cd-efg', '-', 2) -->'cd'
split_to_map(string, entryDelimiter, keyValueDelimiter)	map<varchar, varchar>	entryDelimiter と keyValueDelimiter で文字列を分割した後のmapを返します 例えば：select split_to_map('a:1,b:2', ',', ':') --> {'a':'1','b':'2'}
split_to_map(string, entryDelimiter, keyValueDelimiter, function(K, V1, V2, R))	map<varchar, varchar>	entryDelimiter と keyValueDelimiter で文字列を分割した後のmapを返します。重複するkeyがある場合、functionで指定されたルールに従ってkeyに対応するvalueを返します 例えば：split_to_map('a:1,b:2;a:3',',' , ':', (k, v1, v2) -> v1)-->{'a':'1','b':'2'}
substr(string, start)	varchar	文字列 string の start 位置以降の文字列を返します。位置は1から始まり、start <0の場合は文字列の末尾からカウントします
substr(string, start, length)	varchar	文字列 string の start 位置から length の長さの文字列を返します。位置は1から始まり、start <0の場合は文字列の末尾からカウントします
position(substring IN string)	bigint	string内でsubstringが最初に現れる位置を返します。位置は1から始まります。見つからない場合は0を返します。
strpos(string, substring)	bigint	string内でsubstringが左から右へ最初に現れる位置を返します。位置は1から始まります。見つからない場合は0を返します。 strpos('abcdefg', 'a') -->1
strpos(string, substring, instance)	bigint	string内でsubstringが左から右へinstance回目に現れる位置を返します。instance >0、位置は1から始まります。見つからない場合は0を返します。 例えば：strpos('abcaefg', 'ab',2) -->0

strrpos(string, substring)	bigint	string内でsubstringが右から左へ最初に現れる位置を返します。位置は1から始まります。見つからない場合は0を返します。 例えば：strpos('abcdefg', 'a') -->7
strrpos(string, substring, instance)	bigint	string内でsubstringが右から左へinstance回目に現れる位置を返します。instance >0、位置は1から始まります。見つからない場合は0を返します。 例えば：strpos('abcaefg', 'a',2) -->0
to_utf8(string)	varbinary	文字列をutf8エンコードされたバイナリ数に変換する
from_utf8(binary)	varchar	バイナリ数をutf8エンコードされた文字列に変換し、無効な文字はUnicode文字U+FFFDで置き換えます
from_utf8(binary, replace)	varchar	バイナリ数をutf8エンコードされた文字列に変換し、無効な文字はreplaceで置き換えます

日時関数

関数名	戻り値	関数機能説明
current_date	date	現在の日付を返す
current_time	time	現在のタイムゾーン付きの時間を返す
current_timestamp	timestamp	現在のタイムゾーン付きのタイムスタンプを返す
current_timezone()	varchar	現在のタイムゾーンを返す
date(x)	date	現在の日付を返す
last_day_of_month(x)	date	今月の最終日を返す
from_unixtime(unixtime)	timestamp	unixタイムスタンプを返す from_unixtime(1475996660) -->2016-10-09 15:04:20.000
to_unixtime(timestamp)	double	timestampのunixタイムスタンプを返す
from_unixtime(unixtime, string)	timestamp	unixtimeのタイムスタンプを返し、stringでタイムゾーンを指定します from_unixtime(1617256617,'Asia/Shanghai') -->2021-04-01 13:56:57.000 Asia/Shanghai

from_iso8601_timestamp(string)	timestamp	iso8601形式で文字列をタイムスタンプに変換する
from_iso8601_date(string)	date	iso8601 形式で string を日付に変換します
localtime	time	現在の時刻を返す
localtimestamp	timestamp	現在のタイムスタンプを返す
now()	timestamp	現在のタイムゾーン付きのタイムスタンプを返す
to_iso8601(x)	varchar	xのiso8601形式の文字列を返します。xは日付、タイムスタンプ、またはタイムゾーン付きのタイムスタンプを指定できます
date_trunc(unit, x)	入力と一致	xの時間をunit単位で切り捨てます。unitはsecond、minute、hour、day、week、month、quarter、yearを指定できます。 date_trunc('hour', TIMESTAMP '2020-03-17 02:09:30') --> '2020-03-17 02:00:00'
date_add(unit, value, timestamp)	入力と一致	対応する単位の間隔時間を加減算して新しいタイムスタンプを算出します。value >0 は加算、value < 0 は減算を表します。 unit は millisecond、second、minute、hour、day、week、month、quarter、year です。 例えば：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	指定された単位（ミリ秒、秒、分、時、日、週、月、四半期、年）で2つのタイムスタンプの時間差を返します。 例えば：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	fieldからxの時間を抽出して返します。fieldには次の値を指定できます：year、quarter、month、week、day、dow、doy、yow、hour、minute、second、timezone_hour、timezone_minute、day_of_week、day_of_year、day_of_month 例えば：extract(year from current_date）2023
day(x)	bigint	x が月の何日目であるかを返します
day_of_month(x)	bigint	x が月の何日目であるかを返します

day_of_week(x)	bigint	x が週の何日目であるかを返します（範囲は1～7）
day_of_year(x)	bigint	x が年の何日目であるかを返します
dow(x)	bigint	x が週の何日目であるかを返します。day_of_week(x)と同じです
doy(x)	bigint	x が年の何日目であるかを返します。day_of_year(x)と同じです
hour(x)	bigint	x が一日の何時目かを返します。範囲は0～23です
millisecond(x)	bigint	x が一秒の何ミリ秒目かを返します
minute(x)	bigint	x が一時間の何分目かを返します
month(x)	bigint	x が年の何ヶ月目であるかを返します
quarter(x)	bigint	x が年の第何四半期であるかを返します
second(x)	bigint	x が一分の何秒目かを返します
week(x)	bigint	x が年の何週目であるかを返します
week_of_year(x)	bigint	x が年の何週目であるかを返します
year(x)	bigint	x の年を返します

配列関数

関数名	戻り値	関数機能説明
array_distinct(x)	array	配列から重複する値xを削除します。
array_intersect(x, y)	array	x と y の交差部分から重複しない要素を返します。
array_union(x, y)	array	x と y の和集合から重複しない要素を返します。
array_except(x, y)	array	x に属し、y に属さない重複しない要素を返します。（差集合）
array_join(x, delimiter, null_replacement)	varchar	delimiterを使用してx配列の要素を接続し、null_replacementで配列内の空の値を埋めます。null_replacementはオプションの文字です。
array_max(x)	x	入力配列の最大値を返す
array_min(x)	x	入力配列の最小値を返す

array_position(x, element)	bigint	配列x内で最初にelementが出現する位置（数字）を返します（見つからない場合は0を返します）
array_remove(x, element)	array	配列xからelementと同じすべての要素を削除します
array_sort(x)	array	xのソート結果を返します。xの要素はソート可能でなければなりません。空の要素は返される配列の末尾に配置されます。
array_sort(array(T), function(T, T, int))	array(T)	配列を比較関数functionに基づいてソートした結果を返します 例えば：array_sort(ARRAY [3, 2, 5, 1, 2], (x, y) –> IF(x < y, 1, IF(x = y, 0, –1))) ––>[5, 3, 2, 2, 1]
arrays_overlap(x, y)	ブール値	配列 x と y に共通の非空要素があるかどうかを判断します
cardinality(x)	bigint	配列の基数（サイズ）を返します
concat(array1、array2、…、arrayN)	array	配列array1、array2、...、arrayNを連結します
contains(x, element)	ブール値	配列 x が element を含むかどうかを判断します
element_at（array(E)、index)	E	配列 array の index 位置の要素を返します。index > 0 の場合は左から右に数え、index < 0 の場合は右から左に数えます。
filter(array(T), funciton(T, boolean))	array(T)	関数functionがtrueを返す要素で構成される新しい配列を返します 例えば：filter(array[5, –6, NULL, 7], x –> x > 0); ––> [5, 7]
repeat(element、count)	array	elementをcount回繰り返す
reverse(x)	array	配列xの逆順の配列を返します。
sequence（start, stop, n)	array（bigint)	startからstopまで、ステップnで整数シーケンスを生成します。nはオプションで、指定しない場合は1です。startがstop以下の場合、毎回1ずつ増加し、それ以外の場合は–1ずつ増加します。
shuffle(x)	array	配列の要素をシャッフルします
slice(x, start, length)	array	start位置から長さlengthの配列にスライスした配列xを返します。

transform(array(T), function(T, U))	array(U)	array配列の各要素Tをfunctionで処理した後に生成されるUで構成される配列を返します 例えば：transform(array [5, 6], x -> x + 1); -- [6, 7]

JSON関数

関数名	戻り値	関数機能説明
is_json_scalar(json)	boolean	jsonがjson数値またはjson文字列またはtrueまたはfalseまたはnullであるかどうかを判断します 例えば： is_json_scalar('[1, 2, 3]') -->false
json_array_contains(json, value)	boolean	json（json配列を含む文字列）にvalueが存在するかどうかを判断します。 例えば： json_array_contains('[1, 2, 3]', 2) -->true
json_array_length(json)	bigint	json（json配列を含む文字列）の配列の長さを返します。 例えば： json_array_length('[1, 2, 3]') --> 3
json_extract(json, json_path)	json	json（JSONを含む文字列）に対してJSONPathのような式json_pathを計算し、結果をjsonとして返します。 例えば： json_extract('{"log":{"file":{"path":"/etc/nginx/logs/access.log"},"offset":19991212}}', '$.log.file.path')--> "/etc/nginx/logs/access.log"
json_extract_scalar(json, json_path)	varchar	json_extract()と同様ですが、結果の値をJSON文字列ではなく文字列として返します。json_pathで参照される値は、ブール値、数値、または文字列でなければなりません。 例えば：json_extract_scalar('{"log":{"file":{"path":"/etc/nginx/logs/access.log"},"offset":19991212}}', '$.log.file.path') -->/etc/nginx/logs/access.log
json_format(json)	varchar	入力されたJSON値からシリアライズされたJSONテキストを返します。これはjson_parse()の逆関数です。 例えば：json_format(JSON '[1, 2, 3]') -->[1,2,3]
json_parse(string)	json	入力されたJSONテキストからデシリアライズされたJSON値を返します。これはjson_format()の逆関数です。 例えば：json_parse('[1, 2, 3]') -->[1,2,3]
json_size(json, json_path)	bigint	json_extract()と同様ですが、戻り値のサイズを返します。オブジェクトまたは配列の場合、サイズはメンバーの数であり、スカラー値のサイズはゼロです。 例えば：json_size('{"x": [1, 2, 3]}', '$.x') -->3

集約関数

関数名	戻り値	関数機能説明
arbitrary(x)	入力と一致	存在する場合、xの任意の非NULL値を返します
array_agg(x)	array[x]	入力x要素から作成された配列を返します
avg(x)	double	すべての入力値の平均値（算術平均）を返します
bool_and(boolean) | every(boolean)	boolean	すべての入力値がtrueの場合、trueを返し、それ以外の場合はfalseを返します
bool_or(boolean)	boolean	入力された値のいずれかがtrueの場合、trueを返し、それ以外の場合はfalseを返します
checksum(x)	varbinary	入力された値のチェックサムを返します
count(*)	bigint	行数を返します
count(x)	bigint	空でない入力値の数を返します
count_if(x)	bigint	入力値の中でtrueの値の数を返します。count(CASE WHEN x THEN 1 END)と同等です
geometric_mean(x)	double	幾何平均値を返します
max_by(x, y)	入力 x と一致	x における y の最大値に関連する値を返します
max_by(x, y, n)	array[x]	y の降順に並べ替え、x における y の n 個の最小値に関連する n 個の値を返します
min_by(x, y)	入力 x と一致	x における y の最小値に関連する値を返します
min_by(x, y, n)	array[x]	y の昇順に並べ替え、x における y の n 個の最小値に関連する n 個の値を返します
max(x)	入力と一致	入力中の最大値を返す
max(x, n)	入力 x と一致	入力中の最大の n 個の値を返す
min(x)	入力と一致	入力中の最小値を返す
min(x, n)	入力 x と一致	入力中の最小のn個の値を返す
set_agg(x)	入力と一致	重複しない要素の配列を返す
set_union(array(T))	array(T)	入力の各配列に含まれるすべての異なる値の配列を返す

		例えば：select set_union(elements) FROM (VALUES ARRAY[1, 2, 3], ARRAY[2, 3, 4]) AS t(elements) -->ARRAY[1, 2, 3, 4]
sum(x)	入力と一致	返す
bitwise_and_agg(x)	bigint	すべてのビットのANDを返す
bitwise_or_agg(x)	bigint	すべてのビットのORを返す
histogram(x)		各入力値の出現回数を含むマップを返します
map_agg(key, value)	map(K, V)	key–valueで構成されたmapを返す
map_union(x(K, V))	map(K, V)	すべての入力マッピングの和集合を返します。1つのキーが複数の値に対応する場合、結果セット内のキーに対応する値は複数の値のいずれか1つになります
map_union_sum(x(K, V))	map(K, V)	同じkeyを合計した後の和集合mapを返します。keyに対応するvalueが空の場合、0として計算されます
multimap_agg(key, value)		key–valueで構成されたmapを返します。keyは複数の値に対応できます

ウィンドウ関数

関数名	戻り値	関数機能説明
row_number()	bigint	各行に一意の連続番号を割り当てます
rank()	bigint	ある値が一連の値の中で占める順位を計算します。順位が等しい場合、順位シーケンスに空きが生じます。
percent_rank()	double	ある値が一連の値の中で占めるパーセンテージ順位を計算します。戻り値は0から1の間の小数で表されます。
dense_rank()	bigint	ある値が一連の値の中で占める順位を計算します。ランクが等しい場合、関数rankとは異なり、dense_rankは順位シーケンスに空位を生じさせません。
cume_dist()	bigint	ある値がパーティション内のすべての値に対して占める位置を計算します。
ntile(n)	bigint	ウィンドウパーティションの行をn個のバケットに分割し、行が属するバケット番号を1からnの範囲で返します。
first_value(x)	入力と一致	パーティション内の列の最初のデータの値を返します

last_value(x)	入力と一致	パーティション内の列の最後のデータの値を返します
nth_value(x, n)	入力と一致	ウィンドウの先頭からn行目の値を返します。nは1から始まります。 ignoreNulls=trueの場合、n行目を検索する際にnullをスキップします。それ以外の場合、各行がnにカウントされます。 そのようなn番目の行が存在しない場合（例えば、nが10でウィンドウサイズが10未満の場合）、nullを返します。最初の引数は列名、2番目の引数は前のn番目の行です。
lead(x[, n[, default_value]])	入力と一致	ウィンドウ内の現在の行から下にn行目の値を返します。nのデフォルト値は1で、defaultのデフォルト値はnullです。 n行目の値がnullの場合、nullを返します。 そのようなオフセット行が存在しない場合（例えば、オフセットが1でウィンドウの最後の行に下方向の行がない場合）、デフォルト値を返します。最初の引数は列名、2番目の引数は前のn行目、3番目の引数はデフォルト値です。
lag(x[, n[, default_value]])	入力と一致	ウィンドウ内の現在の行から上にn行目の値を返します。nのデフォルト値は1で、defaultのデフォルト値はnullです。 n行目の値がnullの場合、nullを返します。 そのようなオフセット行が存在しない場合（例えば、オフセットが1でウィンドウの最初の行に上方向の行がない場合）、defaultを返します。最初の引数は列名、2番目の引数は前のn行、3番目の引数はデフォルト値です。

URL関数

関数名	戻り値	関数機能説明
url_extract_host(url)	varchar	urlのホストを返す
url_extract_parameter(url, name)	varchar	url 内の name の最初の値を返します。RFC 1866#section-8.2.1 に従って検索します
url_extract_path(url)	varchar	url のパスを返します
url_extract_port(url)	bigint	url のポートを返します
url_extract_protocol(url)	varchar	url のプロトコルを返します
url_extract_query(url)	varchar	url のクエリ文字列を返します
url_encode(value)	varchar	value を encode エンコードします

url_decode(value)	varchar	encode エンコードされたurlをデコードします

その他の関数

関数名	戻り値	関数機能説明
uuid ( )	uuid	ランダムに生成されたUUIDを返します
cast(value AS type)	type	value を type 型に強制変換します
try_cast(value AS type)	type	value を type 型に変換し、失敗した場合は null を返します
typeof(expr)	varchar	expr の型名を返します 例えば： typeof(cos(2) + 1.5) -->double
regexp_extract(string, pattern)	varchar	string 内で正規表現 pattern に一致する最初の部分文字列を返します 例えば：regexp_extract('1a 2b 14m', '\d+') ->1
regexp_replace(string, pattern)	varchar	string から正規表現 pattern に一致する部分文字列を削除します 例えば：regexp_replace('1a 2b 14m', '\d+[ab] ') --> '14m'
regexp_split(string, pattern)		正規表現 pattern を使用して文字列を分割し、配列を返します。後続の空文字列を保持します 例えば：regexp_split('1a 2b 14m', '\s*[a-z]+\s*'); --> [1, 2, 14,]
regexp_like(string, pattern)	boolean	string に pattern に一致する文字列が含まれているかどうかを判断します

Hive 関数対応表

最終更新日：：2025-12-25 12:00:07

DLCはHive関数を完全にサポートしており、Hiveからデータレイクコンピューティング DLCに簡単にアップグレードし、より強力なデータレイク機能を利用できます。
DLC統一関数とHive関数には一部細かい違いがあり、具体的な関数対応表は以下の通りです。
Hive関数の詳細については、Apache Hive公式サイト LanguageManual UDF を参照してください。

数学関数

Hive数学関数については、Apache Hive公式サイト MathematicalFunctions を参照してください。
DLC数学関数については、Tencent Cloud公式ドキュメント 数学関数 を参照してください。

Hive3.1関数名	関数機能説明	DLC関数名	差異表現	使用参考
round	四捨五入	round	差異なし	select round(1.23);
round	指定された桁数で四捨五入	round	差異なし	select round(1.234,2);
bround	HALF_EVENモードでの四捨五入	bround	DLC関数では、broundに2つのパラメータを渡す必要があります。2番目のパラメータは保持する小数点以下の桁数を指定します。2番目のパラメータを0に設定すると、hiveのbround(double a)と同じ効果があります。	select bround(1.237,0);
bround	HALF_EVENモードでの四捨五入，指定された小数位を保持	bround	差異なし	select bround(1.237,2);
floor	切り捨て	floor	差異なし	select floor(1.23);
ceil, ceiling	切り上げ	ceil,	差異なし	select ceil(1.93);

		ceiling		select ceiling(1.13);
rand	0から1の範囲内のランダムな数値を返します。シード値（seed）を指定すると、安定したランダム数列が得られます	rand	差異なし	select rand(); select rand(5);
exp	e の a 乗	exp	差異なし	select exp(1); select exp(2.4);
ln	自然対数関数	ln	差異なし	select ln(2.4);
log10	10を底とする対数関数	log10	差異なし	select log10(2.4);
log2	2を底とする対数関数	log2	差異なし	select log2(2.4);
log	対数関数	log	差異なし	select log(2,4);
pow, power	冪関数	pow，power	差異なし	select pow(2,4);
sqrt	冪根関数	sqrt	差異なし	select sqrt(4);
bin	二進関数	bin	差異なし	select bin(4);
hex	十六進関数	hex	差異なし	select hex(10);
unhex	16進数を文字列に変換	unhex	DLC Spark関数では結果をデコードする必要があります	select decode(unhex('3141'), 'UTF-8');
conv	進数変換	conv	差異なし	select conv(2,10,2);
abs	絶対値を取る	abs	差異なし	select abs(-1);
pmod	モジュロを取る	pmod	差異なし	select pmod(5,3);
sin	三角関数 sin、a はラジアン	sin	差異なし	select sin(3.14);
asin	三角関数 arc sin、a はラジアン	asin	差異なし	select asin(0.5);

cos	三角関数 cos、a はラジアン	cos	差異なし	select cos(3.14);
acos	三角関数 arc cos、a はラジアン	acos	差異なし	select acos(1);
tan	三角関数 tan、a はラジアン	tan	差異なし	select tan(3.14);
atan	三角関数 arc tan、a はラジアン	atan	差異なし	select atan(1);
degrees	ラジアンから角度へ変換	degrees	差異なし	select degrees(3.14);
radians	角度からラジアンへ変換	radians	差異なし	select radians(180);
positive	aを返す	positive	差異なし	select positive(-1);
negative	-aを返す	negative	差異なし	select negative(1);
sign	aが正数の場合は1.0を返し、aが負数の場合は-1.0を返します。aが0の場合は0.0を返します	sign	差異なし	select sign(1.12);
e	自然定数 e を返します	e	差異なし	select e();
pi	円周率 pi を返します	pi	差異なし	select pi();
factorial	階乗	factorial	差異なし	select factorial(5);
cbrt	立方根	cbrt	差異なし	select cbrt(27);
shiftleft	左シフト	shiftleft	差異なし	select shiftleft(3,1);
shiftright	右シフト	shiftright	差異なし	select shiftright(3,1);
shiftrightunsigned	符号なし右シフト	shiftrightunsigned	差異なし	select shiftrightunsigned(3,1);

greatest	最大値を返す	greatest	パラメータにnull値がある場合、DLC関数はnull値を計算せず、null以外の最大値を返しますが、Hive関数はnullを返します	select greatest(1,2,3.3);
least	最小値を返す	least	パラメータにnull値がある場合、DLC関数はnull値を計算せず、null以外の最小値を返しますが、Hive関数はnullを返します	select least(1,2,3.3);
width_bucket	バケット値、min_value/max_valueに基づいてnum_buckets+1個の同じサイズのバケットを作成し、現在の値が属するバケット番号を返します	width_bucket	差異なし	select width_bucket(10,1,20,2);

集合関数

DLC集合関数とHive集合関数は差異なし。
Hive集合関数については、Apache Hive公式サイト CollectionFunctions を参照してください。
DLC集合関数については、Tencent Cloud公式ドキュメント 集合関数 を参照してください。

Hive3.1関数名	関数機能説明	DLC関数名	差異表現	使用参考
size	map / array のサイズを返す	size	差異なし	select size(str_to_map('a:1,b:2'));
map_keys	マップのキー値リストを返す	map_keys	差異なし	select map_keys(str_to_map('a:1,b:2'));
map_values	map の value 値リストを返す	map_values	差異なし	select map_values(str_to_map('a:1,b:2'));

array_contains	array に value が含まれているかどうか	array_contains	差異なし	select array_contains(split('a,b',','),'a');
sort_array	array の要素を昇順に並べ替える	sort_array	差異なし	select sort_array(split('1,3,2',','));

型変換関数

Hive3.1関数名	関数機能説明	DLC関数名	差異表現	使用参考
binary	入力されたパラメータをバイナリ配列に変換します	binary	• Hive3.1の入力パラメータはstringとbinaryのみサポートしています；DLCはstring、int、long及びbinaryの入力パラメータをサポートしています • Hive3.2の出力はstringとして表示されますが、DLCの出力はバイナリ配列であり、stringに変換されません。	select binary('testString') select binary(1)（HIVE はサポートしていません） select binary(inputCol) from inputTable（HIVE はサポートしていません）
cast	• 入力されたパラメータを指定されたtype型に強制的に変換します。 • 指定された式または値を指定された型に強制的に変換できない場合、エラーが発生します。例えば、文字列をlongに変換する際のエラー：Cast function cannot convert value of type	cast	Hive3.1では強制変換に失敗した場合NULLが返されますが、DLCではエラーが発生します。	select cast('10' as int)select cast(inputCol as int) from inputTable

	VARCHAR(65536) to type LONG			

時間関数

Hive3.1関数名	関数機能説明	DLC対応関数名	差異表現	使用参考
from_unixtime	Unix時間（1970-01-01 00:00:00 UTCからの経過秒数）を指定された形式の文字列に変換する関数。デフォルトの出力時間形式は yyyy-MM-dd HH:mm:ss で、タイムゾーンは現在のシステムで定義されたタイムゾーンです。	from_unixtime	差異なし	select from_unixtime(12458456) select from_unixtime(100, 'yyyyMMdd HH:mm:ssZ')
unix_timestamp	指定された時間、またはデフォルトで現在の時間を指定し、1970-01-01 00:00:00 UTCから指定された時間までの秒数を返します。	unix_timestamp	差異なし	select unix_timestamp() select unix_timestamp('2023-04-12 00:00:00') select unix_timestamp('2023-04-12', 'yyyy-MM-dd')
to_date	指定された時間を指定し、その時間が含まれる日付を取得します。例えば2023-04-12 13:14:20が含まれる日付は2023-04-12です。	to_date	• HIVEのto_date関数は、指定された形式 yyyy-MM-dd HH:mm:ss でのみ入力でき、指定されていない形式の場合は一律	select to_date('2023-04-12 19:41:23') select to_date('20230412 19:41', 'yyyy-MM-dd HH:mm')（HIVE はサポートしていません） select to_date('20230412 19:41', 'yyyyMMdd HH:mm')（HIVE はサポートしていません）

			NULLを返します。 • DLCはデータ形式を指定するためのパラメータを追加でき、入力時間形式をより柔軟にすることができます。	
year	指定された時間を指定し、その時間が含まれる年を取得します。 例えば2023-04-12 13:14:20が含まれる年は2023です	year	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select year('2023-04-12 19:41:25') select year('2023-04')（HIVEはサポートしていません）
quarter	指定された時間を指定し、その時間が含まれる四半期を取得します。 例えば2023-04-12 13:14:20が含まれる四半期は2です	quarter	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-	select quarter('2023-04-12 19:41') select quarter('2023-04')（HIVE はサポートしていません）

			MM形式をサポートできます。	
month	指定された時間を指定し、その時間が含まれる月を取得します。 例えば2023-04-12 13:14:20が含まれる月は4です。	month	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • 時間形式の月が12を超える場合、HIVEは最後の月との差を計算して余りを取得しますが、DLCはNULLを返します。	select month('2023-04-12 19:41') select month('2023-04')（HIVE はサポートしていません）
day/dayofmonth	指定された時間を指定し、その時間が含まれる日を取得します。 例えば2023-04-12 13:14:20が含まれる月の12日目です。	day/dayofmonth	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。	select day('2023')（HIVE はサポートしていません） select dayofmonth('2023-04-12 19:41') select day('2023-04-12 19:41')

			• DLCはyyyyまたはyyyy-MM形式をサポートできます。 • 時間形式の日数がその月の日数を超える場合、HIVEはその月の日数で余りを取得しますが、DLCはNULLを返します。	
hour	指定された時間を指定し、その時間が含まれる時間を取得します。 例えば2023-04-12 13:14:20が含まれる時間の13時間目です。	hour	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • 時間形式の時間が23を超える場合、HIVEは23で余りを取得しますが、DLCはNULLを返します。	select hour('2023-04-12 19:41') select hour('2023-04')（HIVEはサポートしていません）

minute	指定された時間を指定し、その時間が含まれる分を取得します。 例えば2023-04-12 13:14:20が含まれる分は14分目です。	minute	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • 時間形式の分が59を超える場合、HIVEは60で余りを取得しますが、DLCはNULLを返します。	select minute('2023-04-12 00:41') select minute('2023-04')（HIVE はサポートしていません）
second	指定された時間を指定し、その時間が含まれる秒を取得します。 例えば2023-04-12 13:14:20が含まれる秒は20秒目です。	second	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select second('2023-04-12 00:41:24') select second('2023-04')（HIVE はサポートしていません）

			• 時間形式の秒が59を超える場合、HIVEは60で余りを取得しますが、DLCはNULLを返します。	
weekofyear	指定された時間が年間の第何週に当たるかを返します。 例えば2023-04-12 13:14:20は2023年の第15週です。 特に注意が必要なのは、当年の最初の月曜日が第1週の開始と見なされ、日付が当年の最初の月曜日より前の場合、前年の第52週と見なされることです。例えば、2023-01-01は日曜日であるため、weekofyear('2023-01-01')は52、つまり前年の第52週を返し、weekofyear('2023-01-02')は1、つまり当年の第1週を返します。	weekofyear	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select weekofyear('2023-04-12 00:41:25') select weekofyear('2023-04')（HIVE はサポートしていません）

extract	extract(field from source)は入力されたsourceの時間または時間間隔から指定されたfieldフィールドを抽出します	extract	• HIVEとDLCバージョンは現在、精度の調整をサポートしていません。 • DATEまたはTIMESTAMP識別子が明示的に指定されている場合、日付または時刻の形式が一致しないと、DLCはエラーを報告し、HIVEは0を返します。 • HIVEはINTERNALの間隔を計算する際、fieldがsourceに存在しない場合、自動的に剰余を計算しますが、DLCはエラーを報告します。例えば、24か月の間隔から年のフィールドを抽出する場合、HIVEは2年を返し、DLCはフィールドが存在しないというエ	select extract(DAY FROM DATE '2023-04-12') 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) （DLCはサポートしていません） select extract(hour from interval '23 13:23:34.34784' DAY TO SECOND)

			ラーを報告します。	
datediff	startDateからendDateまでの日数を計算します。startDateがendDateより遅い場合、負の数が返されます。 いずれかの入力がNULLの場合、NULLを返します	datediff	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select datediff('2021', '2022-10')（HIVE はサポートしていません） select datediff('2023-04-13', '2022-04-13 11:00:00')
from_utc_timestamp	指定されたUTC標準時間のtimestampを、指定されたタイムゾーンtimeZoneの時間に変換します。 いずれかの入力がNULLの場合、NULLを返します。	from_utc_timestamp	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • Hiveが返すのは時間形式で、文字列表現はyyyy-MM-dd	select from_utc_timestamp('2023-04', 'Asia/Seoul') select from_utc_timestamp('2023-04-12 15:00:00', 'Asia/Shanghai')

			HH:mm:ssです。 • 一方、DLCが返すのはUTC標準表現で、例えば上海タイムゾーンではyyyy-MM-ddTHH:mm:ss+08:00となります。	
to_utc_timestamp	指定されたタイムゾーンtimeZoneの時間のtimestampを、UTC標準時間のtimestampに変換します。 いずれかの入力がNULLの場合、NULLを返します。	to_utc_timestamp	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • Hiveが返すのは時間形式で、文字列表現はyyyy-MM-dd HH:mm:ssです。 • 一方、DLCが返すのはUTC標準表現で、例えば上海タイムゾーンで	select to_utc_timestamp('2023-04-12 15:00:00', 'Asia/Shanghai') select to_utc_timestamp('2023-04', 'Asia/Shanghai')（HIVE はサポートしていません）

			はyyyy-MM-ddTHH:mm:ss+08:00となります。	
date_add	日付の加算、指定されたstartDateの日付にnumDays日を加えた日付を返します。 numDaysは負の数でも構いません。 いずれかの入力がNULLの場合、NULLを返します。	date_add	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select date_add('2023-04-12 15:00:00', 1) select date_add('2023-04', -1)（HIVE はサポートしていません）
date_sub	日付の減算、指定されたstartDateの日付からnumDays日を引いた日付を返します。 numDaysは負の数でも構いません。 いずれかの入力がNULLの場合、NULLを返します。	date_sub	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select date_sub('2023-04-12 15:00:00', 1) select date_sub('2023-04', -1)（HIVE はサポートしていません）
current_date	現在の日付を取得	current_date	差異なし	select current_date()

current_timestamp	現在の時刻を取得	current_timestamp	差異なし	select current_timestamp()
add_months	指定された日付（または時刻）に指定された月数を加算した日付を計算します。 入力がNULLの場合、NULLを返します。	add_months	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select add_months('2023-04-12 15:00:00', 1) select add_months('2023-04', -1)（HIVE はサポートしていません）
last_day	指定された日付が含まれる月の最終日を計算します。 入力がNULLの場合、NULLを返します。	last_day	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	select last_day('2023-04-12 15:00:00') select last_day('2023-04')（HIVE はサポートしていません）
next_day	指定された日付の後の最初のday_of_weekの日付を計算します。 day_of_weekは曜日を表し、月曜日	next_day	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていません	select next_day('2023-04-12 15:00:00') select next_day('2023-04')（HIVE はサポートしていません）

	から日曜日までのオプション値があります。 いずれかの入力がNULLの場合、NULLを返します。		が、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	
trunc	指定された日付をformat（年、四半期、月、週など）で短縮した最初の日の日付を計算します。例えば、該当四半期の最初の日、該当月の最初の日、該当週の最初の日など。	trunc	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。 • HIVEはWEEK週をサポートしていません。	select trunc('2023-04-12 15:00:00', 'MONTH') select trunc('2023-04', 'YEAR')（HIVE はサポートしていません）
-	指定された日付をformat（年、四半期、月、週など）で短縮した最初の日の時間を計算します。例えば、該当四半期の最初の日の0時です。 truncメソッドとの違いは、	date_trunc	HIVEはこの関数をサポートしていません。 trunc関数を参照してください。	select date_trunc('MM', '2023-04-12 15:00:00') select date_trunc('SECOND', '2023-04-12 15:00:60')

	date_truncが時間を返す点です。			
months_between	date1からdate2までの月数を計算します。 date1<date2の場合、負の数が返されます。 なお、計算時には、各月を31日と見積もり、分母を31とします。分子は2つの時間の実際の差をミリ秒単位で正確に算出した値となります。 最終的な計算結果は小数となります。 いずれかの入力パラメータがNULLの場合、NULLを返します。	months_between	• HIVEの入力はyyyyまたはyyyy-MM形式をサポートしていませんが、他の時間または日付形式はすべてサポートしています。 • DLCはyyyyまたはyyyy-MM形式をサポートできます。	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') （HIVE はサポートしていません）
date_format	指定されたフォーマットテンプレート fmt（Datetime Patterns for Formatting and Parsing を参照）に従って、入力された日付をフォーマットします。 いずれかの入力パラメータがNULLの場合、NULLを返します。	date_format	dateまたはTIMESTAMPの形式が正しくない場合、またはfmtテンプレートの形式が正しくない場合、HIVEはNULLを返し、DLCはエラーを報告します。	select date_format('2023-04-12 15:00:00', 'y') select date_format('2023-04-12', 'yyyy-MM-dd HH:mm')

条件関数

Hive3.1関数名	関数機能説明	DLC関数名	差異表現	利用参考

if	条件が真の場合、valueTrueを返し、それ以外の場合はvalueFalseOrNullを返します	if	条件がNULLの場合、HIVEは条件を偽と判断しますが、DLCはエラーを報告します	select if(1<2, 12, 'false')
isnull	aがNULLの場合、trueを返し、それ以外の場合はfalseを返します	isnull	差異なし	select isnull('false')
isnotnull	aがNULLでない場合、trueを返し、それ以外の場合はfalseを返します	isnotnull	差異なし	select isnotnull('false')
nvl	値がNULLの場合、2番目のパラメータを返します	nvl	差異なし	select nvl(NULL, 'false') select nvl(1, 'false')
coalesce	最初のNULLでないパラメータを返します	coalesce	差異なし	select coalesce(NULL, 'false')
case/when	一致判定	case/when	選択したタイプが一致しない場合、HIVEは整数型と文字列型の暗黙的な変換のみをサポートしますが、DLCはさらにboolean型などの暗黙的な変換もサポートします。	select CASE 'c' WHEN 'a' THEN 1 WHEN 'b' THEN 2 ELSE 0 END; select CASE WHEN 1 > 0 THEN 1 WHEN 2 > 0 THEN 2.0 ELSE 1.2 END;
nullif	パラメータ a=b の場合、null を返し、それ以外の場合は a を返します	nullif	HIVEは最初のパラメータがNULLであることをサポートしませんが、DLCは任意のパラメータがNULLであることをサポートできます。	select nullif(1, 'test'); select nullif(NULL, 'test');（HIVEはサポートしていません）
assert_true	「条件」が真でない場合、例外	assert_true	差異なし	select assert_true(1>0)

	をスローします。それ以外の場合はnullを返します。			

文字列関数

Hive3.1関数名	関数機能説明	DLC対応関数名	差異表現	利用参考
ascii	strの最初の文字列の数値を返す	ascii	差異なし	select ascii('222');
base64	パラメータをバイナリからbase64文字列に変換する	base64	差異なし	select base64('tencent');
character_length	UTF-8文字列に含まれる文字数を返す	character_length	差異なし	select char_length(binary('tencent'));
chr	Aと同等のバイナリ文字を返す	chr	差異なし	select chr(65);
concat	引数として渡された文字列またはバイトを順番に連結して生成された文字列またはバイトを返す	concat	差異なし	select concat('Spark', 'SQL');
context_ngrams	コンテキストNグラムが与えられた場合、トークン化された文のセットから上位k個のコンテキストNグラムを返します	context_ngrams	差異なし	-
concat_ws	sepで区切られた文字列を返す	concat_ws	差異なし	select concat_ws(' ', 'tencent', 'dlc');
decode	提供された文字セット（「US-ASCII」、「ISO-8859-1」、「UTF-8」、「UTF-16BE」、「UTF-16LE」、「UTF-16」のいずれか）を使用して、最初の引数を文字列にデコードします。いずれかの引数が空の場合、結果も空になります。	decode	差異なし	select decode(encode('abc', 'utf-8'), 'utf-8');
elt	インデックス番号の文字列を返します。例えば、elt(2,'hello','world')は'world'を返します。Nが1未満または引数	elt	差異なし	select elt(1, 'scala', 'java');

	の数を超える場合、NULLを返します。			
encode	提供された文字セット（「US-ASCII」、「ISO-8859-1」、「UTF-8」、「UTF-16BE」、「UTF-16LE」、「UTF-16」のいずれか）を使用して、最初の引数をバイナリにエンコードします。いずれかの引数がNullの場合、結果もNullになります	encode	差異なし	select encode('abc', 'utf-8');
field	field(val T,val1 T,val2 T,val3 T,...) この関数は、パラメータリストval1,val2,val3,...内のvalのインデックス位置を返し、見つからない場合は0を返します。例えば、field('world','say','hello','world')は3を返します。 この関数はすべての基本データ型をサポートし、パラメータはstr.equals(x)を使用して比較されます。 valがNULLの場合、戻り値は0です。	field	DLC関数では、prestoエンジンのみがサポートされています	select field('world', 'say', 'hello', 'world');
find_in_set	カンマ区切りのリストatr_array 内で指定された文字列 str のインデックス（1から開始）を返します。文字列が見つからない場合、または str にカンマが含まれている場合は0を返します。	find_in_set	差異なし	select find_in_set('ab','abc,b,ab,c,def');
format_number	入力されたパラメータを「#,###,###.##」形式にフォーマットし、D 小数位で四捨五入して、結果を文字列として返します。	format_number	差異なし	select format_number(12332.123456, 4);
get_json_object	jsonオブジェクトを抽出	get_json_object	差異なし	select get_json_object('{"a":"b"}', '$.a');

in_file	in_file(string str, string filename) ファイルfilenameにおいて文字列strが行全体として出現する場合、trueを返します。	非対応	–	–
instr	str において substr が最初に出現するインデックス（1から数える）を返します	instr	差異なし	select instr('SparkSQL', 'SQL');
length	文字列の長さを返します	length	差異なし	select length('Spark SQL ');
locate	pos以降のstr内でsubstrが最初に出現する位置を返します。	locate	差異なし	select locate('bar', 'foobar bar');
lower	Bのすべての文字を小文字に変換して生成された文字列を返します	lower	差異なし	select lower('TENCENT');
lpad	lpad(string str, int len, string pad) 文字列 str の左側を文字 pad で埋めて、長さが len に達するようにします。str の長さが len より大きい場合、戻り値は len 文字に切り捨てられます。埋め文字 pad が空の場合、戻り値は null になります。	lpad	DLC sparksql エンジンでは、pad はオプションパラメータであり、pad のデフォルトはスペースです。	select lpad('hi', 5, '??');
ltrim	入力文字列の先頭（左側）から空白をトリミングして得られた文字列を返します	ltrim	差異なし	select ltrim(' SparkSQL ');
ngrams	マーク付き文の最初のk個のN–gramを返します。	ngrams	差異なし	–
octet_length	文字列データのバイト長またはバイナリデータのバイト数を返します。	octet_length	差異なし	select octet_length('SparkSQL');
parse_url	urlからパスを抽出する	parse_url	差異なし	select parse_url('http://sp

				ark.apache.org/path?query=1', 'HOST');
printf	printf スタイルのフォーマット文字列中のフォーマット文字列を返します	printf	差異なし	select printf("Hello World %d %s", 100, "days");
quote	引用符付き文字列を返します	非対応	–	–
regexp_extract	regexp式に一致し、regexグループインデックスidxに対応するstr内の最初の文字列を抽出します	regexp_extract	差異なし	select regexp_extract('100–200', '(\\d+)–(\\d+)', 1);
regexp_replace	regexp_replace(string INITIAL_STRING, string PATTERN, string REPLACEMENT) INITIAL_STRING内でPATTERNで定義されたJava正規表現構文に一致するすべての部分文字列をREPLACEMENTインスタンスに置き換えた結果の文字列を返します。	regexp_replace	DLC sparksql エンジンには、4番目のオプションパラメータ「position」を追加できます。positionは正の整数リテラルで、文字列内の検索開始位置を表します。デフォルト値は1です。	select regexp_replace('100–200', '(\\d+)', 'num');
repeat	指定された文字列をn回繰り返した文字列を返します	repeat	差異なし	select repeat('123', 2);
replace	replace(string A, string OLD, string NEW) 文字列A内で重複しないすべてのOLDの出現箇所をNEWに置き換えた結果の文字列を返します。	replace	sparksql エンジンの3番目のパラメータ NEW はオプション	select replace('ABCabc', 'abc', 'DEF');

			で、デフォルトは空です	
reverse	reverse(string A) 反転した文字列を返します。	reverse	sparksqlエンジンのパラメータは配列でも可能です	select reverse('Spark SQL');
rpad	rpad(string str, int len, string pad) 文字列 str の右側を文字 pad で埋めて、長さが len に達するようにします。str の長さが len より大きい場合、戻り値は len 文字に切り捨てられます。埋め文字 pad が空の場合、戻り値は null になります。	rpad	sparksqlエンジンでは、padはオプションパラメータであり、pad のデフォルトはスペースです。	select rpad('hi', 5, '??');
rtrim	入力文字列の末尾（右側）から空白をトリミングして得られた文字列を返します	rtrim	差異なし	select rtrim(' SparkSQL ');
sentences	入力文字列 str を単語の配列に分割します	sentences	差異なし	select sentences('Hi there! Good morning.');
space	n個のスペースで構成される文字列を返します	space	差異なし	select concat(space(2), '1');
split	split(string str, string pat) 正規表現 pat（pat は正規表現です）を中心に文字列 str を分割します。	split	DLC sparksqlエンジンには、3番目のオプションパラメータ「limit」があります。limit は返される配列の	select split('oneAtwoBthreeC', '[ABC]');

			長さを制限するために使用できます。	
str_to_map	入力パラメータを区切り文字でキー/値ペアに分割してマップを作成します。	str_to_map	差異なし	select str_to_map('a:1,b:2,c:3', ',', ':');
substr	posから始まり長さlenのstr部分文字列、またはposから始まり長さlenのバイト配列スライスを返します	substr	差異なし	select substr('Spark SQL', 5);
substring	posから始まり長さlenのstr部分文字列、またはposから始まり長さlenのバイト配列スライスを返します	substring	差異なし	select substring('Spark SQL', 5);
substring_index	delimの出現回数countに基づいて、strから部分文字列を返します。countが正数の場合、最後の区切り文字の左側のすべてを返します（左から数えて）。countが負数の場合、最後の区切り文字の右側のすべてを返します（右から数えて）。この関数はdelimのマッチングにおいて大文字と小文字を区別します。	substring_index	差異なし	select substring_index('cloud.tencent.com', '.', 2);
translate	from文字列の文字をto文字列の対応する文字に置き換えることで、input文字列を変換します	translate	差異なし	select translate('AaBbCc', 'abc', '123');
trim	入力文字列の両端から空白をトリミングして得られた文字列を返します	trim	DLC SparkSQLエンジンは、より複雑なトリミング式をサポートしています	select trim(' SparkSQL ');

unbase64	strをbase64文字列からバイナリに変換する	unbase64	差異なし	select unbase64('U3Bhcmsgsql');
upper	strのすべての文字を大文字に変更して返します	upper	差異なし	select upper('tencent');
ucase	strのすべての文字を大文字に変更して返します	ucase	差異なし	select ucase('SparkSQL');
initcap	各単語の最初の文字を大文字に変更し、他のすべての文字は小文字にします。	initcap	差異なし	select initcap('sPark sql');
levenshtein(string A, string B)	2つの与えられた文字列間のレーベンシュタイン距離を返します	levenshtein	差異なし	select levenshtein('kitten', 'sitting');
soundex(string A)	文字列のSoundexエンコーディングを返す	soundex	差異なし	select soundex('Miller');

データマスキング関数

Hive3.1関数名	関数機能説明	DLC対応関数名	差異表現	使用参考
mask	mask(string str[, string upper[, string lower[, string number]]]) str のマスクバージョンを返します。	非対応	–	–
mask_first_n	mask_first_n(string str[, int n]) str のマスクバージョンを返します。最初の n 個の値はマスクに置き換えられます。	非対応	–	–
mask_last_n	mask_last_n(string str[, int n]) str のマスクバージョンを返します。最後の n 個の値はマスクに置き換えられます。	非対応	–	–
mask_show_first_n	mask_show_first_n(string str[, int n]) str のマスクバージョンを返します。最初の n 個の文字はマスクに置き換えられずに表示されます。	非対応	–	–

mask_show_last_n	mask_show_last_n(string str[, int n]) str のマスクバージョンを返します。最初の n 個の文字はマスクに置き換えられずに表示されます。	非対応	–	–
mask_hash	mask_hash(string|char|varchar str) str に基づいてハッシュ値を返します。	非対応	–	–

その他の関数

Hive3.1関数名	関数機能説明	DLC対応関数名	差異表現	使用参考
java_method	反映、映す、反射	非対応	–	–
reflect	リフレクションを使用してパラメータシグネチャをマッチングさせ、Javaメソッドを呼び出します	非対応	–	–
hash	パラメータのハッシュ値を返す	hash	異なるエンジンでは計算方法が異なるため、異なる結果が得られる可能性があります	select hash('tencent', array(123), 2);
current_user	現在のユーザーを返す	current_user	差異なし	select current_user();
logged_in_user	セッション状態から現在のユーザー名を返します。これはHiveに接続する際に提供されたユーザー名です。	非対応	–	–
current_database	現在のデータベース名を返します	非対応	–	–
md5	MD5 128ビットチェックサムを16進数文字列形式で返します	md5	差異なし	select md5('tencent');

sha1	入力パラメータのsha1ハッシュ値を16進数文字列形式で返します	sha1	差異なし	select sha1('tencent');
sha	入力パラメータのsha1ハッシュ値を16進数文字列形式で返します	sha	差異なし	select sha('tencent');
crc32	CRC32アルゴリズムを使用して式の巡回冗長検査値を計算します	crc32	差異なし	select crc32('tencent');
sha2	expr の SHA-2 ファミリのチェックサムを16進数文字列形式で返します。SHA-224、SHA-256、SHA-384、SHA-512 をサポートしています。ビット長0は256と同等です。	sha2	差異なし	select sha2('tencent', 256);
aes_encrypt	aes_encrypt(input string/binary, key string/binary) aes暗号化を使用します	aes_encrypt	DLCのsparksqlエンジンはこの関数をサポートしていません	select hex(aes_encrypt('tencent', '0000111122223333'));
aes_decrypt	aes_decrypt(input binary, key string/binary) AES復号化を使用します	aes_decrypt	DLCのsparksqlエンジンはこの関数をサポートしていません	select aes_decrypt(unhex('B99B99CE3359A736DBB9811ED8815C01'), '0000111122223333');
version	version() エンジンバージョンを返す	version	sparksqlはsparkバージョンを返し、prestoはhiveバージョンを返します	select version();
surrogate_key	surrogate_key([write_id_bits, task_id_bits]) テーブルにデータを挿入する際に、自動的に数値IDを生成します。ACIDまたは挿入専用テーブル	非対応	-	-

	のデフォルト値としてのみ使用できます。			

標準 Spark 構文概要

最終更新日：：2026-01-16 17:18:09

標準 Spark エンジンのカーネルは Spark 3.2 バージョンを基に独自開発されており、Spark のネイティブ構文と動作に互換性があります。詳細な構文については Spark 構文ドキュメントを参照してください。

DDL構文

データベース関連構文

用途	構文
新しいデータベース	CREATE DATABASE
そのメタデータで定義されているすべてのデータベースを表示します	SHOW DATABASES
データベースのプロパティを表示します	DESCRIBE DATABASE
データベースのプロパティを変更します	ALTER DATABASE SET DBPROPERTIES
データベースを削除します	DROP DATABASE

データテーブル関連構文

用途	構文
新しいデータテーブル	CREATE TABLE
データテーブルの作成情報を照会	SHOW CREATE TABLE
テーブル属性を照会	SHOW TBLPROPERTIES
データベース内のすべてのテーブルを照会	SHOW TABLES
データテーブルの列情報及びメタデータ情報を表示する	DESCRIBE TABLE
データテーブルに列を追加する	ALTER TABLE ADD COLUMNS
データテーブルに列を追加する	ALTER TABLE ADD COLUMN AFTER/FIRST

フィールド名を変更する	ALTER TABLE ... RENAME COLUMN
データテーブルの特定のフィールドを削除する	ALTER TABLE DROP COLUMN
データテーブルにパーティション情報を追加する	ALTER TABLE ADD PARTITION
テーブルのパーティションを一覧表示する	SHOW PARTITIONS
データテーブルのパーティション情報を削除する	ALTER TABLE DROP PARTITION
Iceberg テーブルにパーティションフィールドを追加する	ALTER TABLE ADD PARTITION FIELD
Icebergテーブルのパーティションフィールドを削除	ALTER TABLE DROP PARTITION FIELD
テーブル属性の変更	ALTER TABLE SET TBLPROPERTIES
テーブルストレージの場所の変更	ALTER TABLE SET LOCATION
テーブルデータの並べ替え方法を変更	ALTER TABLE ... WRITE ORDERED BY
パーティションテーブルの割り当て戦略を変更	ALTER TABLE ... WRITE DISTRIBUTED BY PARTITION
パーティション情報を更新（Icebergテーブルv2バージョンではサポートされていません）	MSCK REPAIR TABLE
メタデータテーブルを削除	DROP TABLE
SQLの論理または物理プランを表示する	EXPLAIN
テーブルストアドプロシージャを呼び出す	CALL STATEMENT

ビュー関連の構文

用途	構文
select結果をビューとして作成	CREATE VIEW AS

データベース内のビューをクエリする	SHOW VIEWS
ビューの列情報を表示する	DESCRIBE VIEW
ビューの作成ステートメントを表示する	SHOW CREATE VIEW
ビューの列情報を表示する	SHOW COLUMNS IN VIEW
ビューの名前を変更する	ALTER VIEW RENAME TO
ビューの属性を変更する	ALTER VIEW SET TBLPROPERTIES
ビューを削除する	DROP VIEW

関数関連の構文

用途	
関数を作成する	CREATE FUNCTION
関数を作成する構文を確認する	SHOW FUNCTION
関数を削除します	DROP FUNCTION

DML構文

用途	
一行のデータを挿入する	INSERT STATEMENT
一行のデータを置き換える	INSERT OVERWRITE
行レベルのデータ更新操作で、INSERT OVERWRITE操作の代わりに使用できます	MERGE INTO
Iceberg テーブルのメタデータクエリ	TABLE METADATA
クエリ結果をデータテーブルに挿入する	INSERT INTO
Iceberg テーブルのデータを削除する	DELETE STATEMENT
指定された行を更新する	UPDATE

DQL 構文

用途	
データクエリ	SELECT STATEMENT

標準 Presto 構文の概要

最終更新日：：2025-12-25 12:00:07

標準 Presto エンジンのコアは Presto 0.242 をベースに独自開発されており、Presto のネイティブ構文と動作に互換性があり、対話型クエリ分析に適しています。詳細な構文については、Presto 構文ドキュメントを参照してください。

用途	構文	対応可能ですか	備考
関数定義を変更	ALTER FUNCTION	いいえ	
スキーマの名前を変更	ALTER SCHEMA	はい	
データテーブルを変更	ALTER TABLE	はい	iceberg テーブルのみサポートされています。 実行時には、対象テーブルのicebergカタログを三段式で指定するか、先に「use iceberg.dbname」を実行する必要があります
テーブルの統計情報を表示	ANALYZE	はい	hive テーブルのみサポートされています。
現在のトランザクションを送信	COMMIT	いいえ	
関数作成	CREATE FUNCTION	いいえ	
ロール作成	CREATE ROLE	いいえ	
スキーマを作成	CREATE SCHEMA	いいえ	
データテーブルを作成	CREATE TABLE	いいえ	
SELECTを使用してテーブルを作成	CREATE TABLE AS	いいえ	

ビューを作成する	CREATE VIEW	いいえ	
PREPAREを削除する	DEALLOCATE PREPARE	いいえ	
データを削除する	DELETE	いいえ	
テーブルの列情報を表示	DESCRIBE	はい	
PREPARE入力情報を表示	DESCRIBE INPUT	いいえ	
PREPARE出力情報を表示	DESCRIBE OUTPUT	いいえ	
関数削除	DROP FUNCTION	いいえ	
ロール削除	DROP ROLE	いいえ	
スキーマを削除	DROP SCHEMA	いいえ	
データテーブルを削除する	DROP TABLE	いいえ	
ビューを削除する	DROP VIEW	いいえ	
PREPAREを実行する	EXECUTE	はい	
SQLの論理または物理プランを表示する	EXPLAIN	はい	
SQLを実行し、実行プランを表示する	EXPLAIN ANALYZE	はい	
ライセンス	GRANT	いいえ	
指定されたオブジェクトにロールをライセンスする	GRANT ROLE	いいえ	

データを挿入する	INSERT	はい	icebergテーブルの場合、実行時に三段式で対象テーブルのicebergカタログを指定するか、先に「use iceberg.dbname」を実行する必要があります
PREPAREを作成する	PREPARE	はい	
指定したSESSIONをデフォルト値に戻す	RESET SESSION	はい	
権限付与の取り消し	REVOKE	いいえ	
付与された役割をキャンセル	REVOKE ROLES	いいえ	
ロールバックトランザクション	ROLLBACK	いいえ	
データを検索する	SELECT	はい	
ロール設定	SET ROLE	いいえ	
指定されたSESSIONの値を設定	SET SESSION	はい	
CATALOGリストを表示	SHOW CATALOGS	はい	
テーブルの列情報を表示	SHOW COLUMNS	はい	
関数情報を表示	SHOW CREATE FUNCTION	いいえ	
テーブル作成情報を表示	SHOW CREATE TABLE	はい	
ビュー作成情報を表示	SHOW CREATE VIEW	いいえ	
関数リストを表示	SHOW FUNCTIONS	はい	
指定ユーザーの権限を表示	SHOW GRANTS	いいえ	

権限付与されたロールリストを表示	SHOW ROLE GRANTS	いいえ	
ロールリストを表示	SHOW ROLES	いいえ	
スキーマリストを表示	SHOW SCHEMAS	はい	
セッションリストを表示	SHOW SESSION	はい	
テーブルの統計情報を表示	SHOW STATS	はい	
テーブルリストを表示	SHOW TABLES	はい	
トランザクションを開始	START TRANSACTION	いいえ	
テーブルのすべての内容を削除	TRUNCATE	いいえ	
テーブルの内容を更新	UPDATE	いいえ	
デフォルトのSCHEMAまたはデータベースを指定	USE	はい	
インライン表を定義	VALUES	はい	

予約語

最終更新日：：2025-12-25 12:00:07

注意事項

以下の識別子がデータベース名、テーブル名、列名、関数名、ビュー名などの識別子として予約語である場合、デフォルトのエスケープ文字を追加する必要があります：

識別子：` バッククォート

例えば：hour -> `hour`

```
create table `hour` (
id string,
`asc` int
)
```

hour、asc はキーワードです。このような列を作成する必要がある場合は、バッククォートを付けて作成する必要があります。

主要な予約語

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