Alibaba Cloud AnalyticDB for MySQL

System Functions

Issue: 20200709

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Document conventions

Style	Description	Example
	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 **Danger:** Resetting will result in the loss of user configuration data.
	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 **Notice:** If the weight is set to 0, the server no longer receives new requests.
	A note indicates supplemental instructions, best practices, tips, and other content.	 **Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings** > **Network** > **Set network type**.
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click **OK**.
Courier font	Courier font is used for commands.	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`

Style	Description	Example
{} or {a|b}	This format is used for a required value, where only one item can be selected.	switch {active|stand}

Contents

1 Geometry functions

AnalyticDB for MySQL supports the following geometry functions: ST_Point or Point, ST_AsText, ST_GeometryFromText or ST_GeomFromText, ST_Distance, and ST_Distance_Sphere. This topic describes how to use geometry functions.

ST_Point or Point

```
ST_Point(x, y)
```

- Description: This function uses the result of x/y as a coordinate to construct a point value. x and y are two numerical values of the DOUBLE type.
- Return value type: GEOMETRY. If you execute the SELECT statement to query results of this function, the returned results will be nonsensical.
- Example:

```
SELECT ST_Point(1,1);
+----------------------+
| ST_Point(1,1)       |
+----------------------+
|      #?     #?  |
```

ST_AsText

```
ST_AsText(g)
```

- Description: This function returns g in the Well-Known Text (WKT) format.
- Example:

```
SELECT ST_AsText(ST_Point(1,1));
+--------------------------+
| ST_AsText(ST_Point(1,1)) |
+--------------------------+
| POINT (1 1)              |
```

ST_GeometryFromText or ST_GeomFromText

```
ST_GeometryFromText(wkt)
```

- Description: This function uses a WKT-formatted string to construct a value of the GEOMETRY type.
- Return value type: GEOMETRY. If you execute the SELECT statement to query results of this function, the returned results will be nonsensical.

- Example:

```
SELECT ST_GeometryFromText('Point(1 1)');
+----------------------------------+
| ST_GeometryFromText('Point(1 1)') |
+----------------------------------+
|      #?     #?              |
```

```
SELECT ST_AsText(ST_GeometryFromText('Point(1 1)'));
+-------------------------------------------+
| ST_AsText(ST_GeometryFromText('Point(1 1)')) |
+-------------------------------------------+
| POINT (1 1)                               |
```

ST_Distance

```
ST_Distance(g1, g2)
```

- Description: This function returns the linear distance between g1 and g2.
- Return value type: DOUBLE.
- Example:

```
SELECT ST_Distance(ST_Point(1,1), ST_Point(2,2));
+-----------------------------------------+
| ST_Distance(ST_Point(1,1), ST_Point(2,2)) |
+-----------------------------------------+
|                  1.4142135623730951 |
```

ST_Distance_Sphere

```
ST_Distance_Sphere(g1, g2 [, radius])
```

- Description: This function returns the spherical distance between g1 and g2. You can specify the sphere radius. The default radius is 6,370,986 meters.
- Return value type: DOUBLE.
- Example:

```
SELECT ST_Distance_Sphere(point(1,1), point(2,2));
+-----------------------------------------+
| ST_Distance_Sphere(point(1,1), point(2,2)) |
+-----------------------------------------+
|                   157225.08654191086 |
```

```
SELECT ST_Distance_Sphere(point(1,1), point(2,2), 6370986);
+-----------------------------------------------------+
| ST_Distance_Sphere(point(1,1), point(2,2), 6370986) |
+-----------------------------------------------------+
|                          157225.08654191086 |
```

2 Aggregate functions

This topic describes the aggregate functions used in AnalyticDB for MySQL.

- AVG: calculates the average value.
- BIT_AND: returns the result of bitwise AND operations for all bits of the argument.
- BIT_OR: returns the result of bitwise OR operations for all bits of the argument.
- BIT_XOR: returns the result of bitwise XOR operations for all bits of the argument.
- COUNT: calculates the number of records.
- MAX: calculates the maximum value.
- MIN: calculates the minimum value.
- STD/STDDEV: returns the sample standard deviation of all input values.
- STDDEV_POP: returns the population standard deviation of all input values.
- STDDEV_SAMP: returns the population standard deviation for a group of integers, decimals, or floating-point numbers.
- SUM: calculates the sum of all input values.
- VAR_POP: returns the population variance for a group of integers, decimals, or floating-point numbers.
- VAR_SAMP: returns the sample variance for a group of integers, decimals, or floating-point numbers.
- VARIANCE: returns the population variance for a group of integers, decimals, or floating-point numbers.

The aggregate functions in this topic use the `testtable` table as test data.

```
create table testtable(a int) distributed by hash(a);
```

```
insert into testtable values (1),(2),(3);
```

AVG

```
avg(bigint x)
avg(double x)
avg(float x)
```

- Description: This function calculates the average value.
- Return value type: DOUBLE.

- Example:

```
select avg(a) from testtable;
+--------+
| avg(a) |
+--------+
|   2.0 |
```

BIT_AND

```
bit_and(float x)
bit_and(bigint x)
bit_and(double x)
```

- Description: This function returns the result of bitwise AND operations for all bits of the argument.
- Return value type: BIGINT.
- Example:

```
select bit_and(a) from testtable;
+------------+
| bit_and(a) |
+------------+
|          0 |
```

BIT_OR

```
bit_or(float x)
bit_or(bigint x)
bit_or(double x)
```

- Description: This function returns the result of bitwise OR operations for all bits of the argument.
- Return value type: BIGINT.
- Example:

```
select bit_or(a) from testtable;
+---------------------------+
|         bit_or(a)         |
+---------------------------+
|          3                |
```

BIT_XOR

```
bit_xor(double x)
bit_xor(bigint x)
bit_xor(float x)
```

- Description: This function returns the result of bitwise XOR operations for all bits of the argument.

- Return value type: BIGINT.
- Example:

```
select bit_xor(a) from testtable;
+---------------------------+
|       bit_xor(a)       |
+---------------------------+
|       0              |
```

COUNT

```
count([distinct|all] value x)
```

- Description: This function counts the number of records.

  distinct and all specify whether to exclude duplicate records in the counting process. The default value is all, which indicates that all records are counted. If distinct is specified, only records with distinct values are counted.
- Return value type: BIGINT.
- Example:

```
select count(distinct a) from testtable;
+------------------+
| count(DISTINCT a) |
+------------------+
|              3 |
```

MAX

```
max(value x)
```

- Description: This function calculates the maximum value.

  value can be of any data type except BOOLEAN.

  If the value for a row in the specified column is null, this row is ignored.
- Return value type: LONG.
- Example:

```
select max(a) from testtable;
+--------+
| max(a) |
+--------+
```

```
|     3 |
```

MIN

```
min(value x)
```

- Description: This function calculates the minimum value.

  `value` can be of any data type except `BOOLEAN`.

  If the value for a row in the specified column is `null`, this row is ignored.

- Return value type: LONG.
- Example:

```
select min(a) from testtable;
+--------+
| min(a) |
+--------+
|     1 |
```

STD/STDDEV

```
std(double x)
std(bigint x)
stddev(double x)
stddev(bigint x)
```

- Description: This function returns the sample standard deviation of all input values.
- Return value type: DOUBLE.
- Example:

```
select std(a) from testtable;
+-------------------+
| std(a)            |
+-------------------+
| 0.816496580927726 |
```

STDDEV_POP

```
stddev_pop(double x)
stddev_pop(bigint x)
```

- Description: This function returns the population standard deviation of all input values.
- Return value type: DOUBLE.
- Example:

```
select stddev_pop(a) from testtable;
+---------------------------+
|   stddev_pop(a)           |
+---------------------------+
```

```
|   0.816496580927726      |
```

STDDEV_SAMP

```
stddev_samp(double x)
stddev_samp(bigint x)
```

- Description: This function returns the population standard deviation for a group of integers, decimals, or floating-point numbers.
- Return value type: DOUBLE.
- Example:

```
select stddev_samp(a) from testtable;
+---------------------------+
|    stddev_samp(a)       |
+---------------------------+
|       1.0             |
```

SUM

```
sum(double x)
sum(float x)
sum(bigint x)
```

- Description: This function calculates the sum of all input values.
- Return value type: BIGINT.
- Example:

```
select sum(a) from testtable;
+---------------------------+
|       sum(a)          |
+---------------------------+
|        6              |
```

VAR_POP

```
var_pop(double x)
var_pop(bigint x)
```

- Description: This function returns the population variance for a group of integers, decimals, or floating-point numbers. You can also use the VARIANCE() function, which has the same meaning as the VAR_POP function. However, the VARIANCE() function is not a standard SQL function. If no matches are found, VAR_POP() returns the value NULL.
- Return value type: DOUBLE.
- Example:

```
select var_pop(a)  from testtable;
+---------------------------+
|       var_pop(a)        |
```

```
+---------------------------+
|  0.6666666666666666       |
```

VAR_SAMP

```
var_samp(double x)
var_samp(bigint x)
```

- Description: This function returns the sample variance for a group of integers, decimals, or floating-point numbers.
- Return value type: DOUBLE.
- Example:

```
select var_samp(a)  from testtable;
+---------------------------+
|       var_samp(a)         |
+---------------------------+
|          1.0              |
```

VARIANCE

```
variance(double x)
variance(bigint x)
```

- Description: This function returns the population standard variance for a group of integers, decimals, or floating-point numbers. The VARIANCE() function is an extension to standard SQL and can be replaced by the standard SQL function VAR_POP(). If no matches are found, VARIANCE() returns the value NULL.
- Return value type: DOUBLE.
- Example:

```
select variance(a)  from testtable;
+---------------------------+
|       variance(a)         |
+---------------------------+
|   0.6666666666666666      |
```

3 Arithmetic operators

AnalyticDB for MySQL supports the following arithmetic operators.

+	Addition.
-	Subtraction.
*	Multiplication.
/	Division.
DIV	Integer division.
% or MOD	Modulo.
-	Changes the sign of the argument.

+

- Description: This operator is used for addition.
- Example:

```
select 3+5;
+-------+
| _col0 |
+-------+
|     8 |
```

```
select 3+2.9875;
+--------+
| _col0  |
+--------+
| 5.9875 |
```

-

- Description: This operator is used for subtraction.
- Example:

```
select 3-5;
+-------+
| _col0 |
+-------+
|    -2 |
```

```
select 3-1.5;
+-----------+
| _col0     |
+-----------+
```

```
| 1.5      |
```

*****

- Description: This operator is used for multiplication.
- Example:

```
select 3*pi();
+------------------+
| _col0            |
+------------------+
| 9.42477796076938 |
```

/

- Description: This operator is used for division.
- Example:

```
select 3/pi();
+-------------------+
| _col0             |
+-------------------+
| 0.954929658551372 |
```

DIV

- Description: This operator is used for division. The decimal part of the result is discarded.
- Example:

```
select 3 div pi();
+-------+
| _col0 |
+-------+
|     0 |
```

```
select 33 div 2;
+-------+
| _col0 |
+-------+
|    16 |
```

% or MOD

- Description: This operator returns the remainder of one argument divided by the other argument.
- Example:

```
select 3 mod pi();
+-------+
| _col0 |
+-------+
```

```
|  3.0 |
```

```
select 33 % 2;
+-------+
| _col0 |
+-------+
|    1 |
```

-

- Description: This operator converts a positive number to a negative number or a negative number to a positive number.
- Example:

```
select - 2;
+-------+
| _col0 |
+-------+
|   -2 |
```

```
select - 2;
+-------+
| _col0 |
+-------+
|   -2 |
```

4 Bit functions and operators

AnalyticDB for MySQL supports the following bit functions and operators.

- BIT_COUNT: This function converts an argument to a binary value, and then returns the number of bits that are set to 1 in the value.
- &: bitwise AND.
- ~: inverts all bits.
- |: bitwise OR.
- ^: bitwise XOR.
- >>(BITWISE_RIGHT_SHIFT): shifts a value to the right.
- <<(BITWISE_LEFT_SHIFT): shifts a value to the left.

BIT_COUNT

```
bit_count(bigint x)
bit_count(double x)
bit_count(varchar x)
```

- Description: converts an argument to a binary value, and then returns the number of bits that are set to `1` in the value.
- Return value type: BIGINT.
- Example:

```
select bit_count(2);
+--------------+
| bit_count(2) |
+--------------+
|            1 |
```

```
select bit_count(pi());
+-----------------+
| bit_count(pi()) |
+-----------------+
|               2 |
```

```
select bit_count('123');
+------------------+|
bit_count('123') |
+------------------+
|               6 |
```

&

- Description: This function is used for bitwise `AND`.
- Return value type: BIGINT.

- Example:

```
 select 12 & 15;
+--------------------+
| bitwise_and(12, 15) |
+--------------------+
|                 12 |
```

~

- Description: This function inverts all bits.
- Return value type: BIGINT.
- Example:

```
select 2 & ~1;
+--------------------------------+
| bitwise_and(2, bitwise_not(1)) |
+--------------------------------+
|                              2 |
```

|

- Description: This function is used for bitwise OR.
- Return value type: BIGINT.
- Example:

```
select 29 | 15;
+--------------------+
| bitwise_or(29, 15) |
+--------------------+
|                 31
```

^

- Description: This function is used for bitwise XOR.
- Return value type: BIGINT.
- Example:

```
select 1 ^ 10;
+--------------------+
| bitwise_xor(1, 10) |
+--------------------+
|                 11 |
```

>>(BITWISE_RIGHT_SHIFT)

```
bitwise_right_shift(double x, double y)
bitwise_right_shift(varchar x, varchar y)
bitwise_right_shift(bigint x, bigint y)
```

- Description: This function shifts a value to the right.

- Return value type: BIGINT.
- Example:

```
select 3 >> 2;
+--------------------------+
| bitwise_right_shift(3, 2) |
+--------------------------+
|                        0 |
```

```
select 3.4 >> 23.2;
+-------------------------------+
| bitwise_right_shift(3.4, 23.2) |
+-------------------------------+
|                             0 |
```

<<(BITWISE_LEFT_SHIFT)

```
bitwise_left_shift(double x, double y)
bitwise_left_shift(varchar x, varchar y)
bitwise_left_shift(bigint x, bigint y)
```

- Description: This function shifts a value to the left.
- Return value type: BIGINT.
- Example:

```
SELECT 3 << 2;
+--------------------------+
| bitwise_left_shift(3, 2) |
+--------------------------+
|                       12 |
```

```
select '3' << '2';
+-----------------------------+
| bitwise_left_shift('3', '2') |
+-----------------------------+
|                          12 |
```

```
select 3.4 << 23.2;
+-------------------------------+
| bitwise_left_shift(3.4, 23.2) |
+-------------------------------+
|                      25165824 |
```

5 Control flow functions

This topic describes the control flow functions of AnalyticDB for MySQL.

- CASE
- IF
- IFNULL
- NULLIF

The data from the conditiontest table is used in the examples for the control flow functions in this topic.

```
create table conditiontest(a int) distributed by hash(a);
```

```
insert into conditiontest values (1),(2),(3);
```

```
SELECT * FROM conditiontest;
+---+
| a |
+---+
| 2 |
| 1 |
| 3 |
```

CASE

```
CASE expression
   WHEN value THEN result
   [ WHEN ... ]
   [ ELSE result ]
   END
```

- Description: The simple CASE expression sequentially compares expression to value in the WHEN clauses until it finds a match. If a match is found, result of the corresponding THEN clause is returned. Otherwise, result of the ELSE clause is returned.
- Example:

```
SELECT a,
     CASE a
     WHEN 1 THEN 'one'
     WHEN 2 THEN 'two'
     ELSE 'three'
     END as caseresult
FROM conditiontest;
+---+------------+
| a | caseresult |
+---+------------+
| 2 | two        |
| 1 | one        |
```

```
| 3 | three     |
```

```
CASE
   WHEN condition THEN result
   [ WHEN ... ]
   [ ELSE result ]
   END
```

- Description: The advanced CASE expression sequentially searches for condition in the WHEN clauses until it finds the first condition that is TRUE. If a match is found, result of the corresponding THEN clause is returned. Otherwise, result of the ELSE clause is returned.
- Example:

```
SELECT a,
     CASE a
     WHEN a=1 THEN 'one1'
     WHEN a=2 THEN 'two2'
     ELSE 'three3'
     END as caseresult
FROM conditiontest;
+---+------------+
| a | caseresult |
+---+------------+
| 1 | one1       |
| 3 | three3     |
| 2 | three3     |
```

IF

```
if(condition, true_value)
```

- Description: If the condition is true, true_value is returned. Otherwise, null is returned.
- Example:

```
SELECT IF((2+3)>4,5);
+-------+
| _col0 |
+-------+
|    5 |
```

```
if(condition, true_value, false_value)
```

- Description: If the condition is true, true_value is returned. Otherwise, false_value is returned.
- Example:

```
SELECT IF((2+3)<5,5,6)
+-------+
| _col0 |
+-------+
```

```
|    6 |
```

IFNULL

```
IFNULL(expr1,expr2)
```

- Description: If `expr1` is not null, `expr1` is returned. Otherwise, `expr2` is returned.
- Example:

```
SELECT IFNULL(NULL,2);
+-------+
| _col0 |
+-------+
|    2 |
SELECT IFNULL(1,0);
+-------+
| _col0 |
+-------+
|    1 |
```

NULLIF

```
NULLIF(expr1,expr2)
```

- Description: If `expr1` is equal to `expr2`, `null` is returned. Otherwise, `expr1` is returned.
- Example:

```
SELECT NULLIF (2,1);
+-------+
| _col0 |
+-------+
|    2 |
SELECT NULLIF (2,2);
+-------+
| _col0 |
+-------+
| NULL  |
```

6 Date and time functions

This topic describes the date and time functions in AnalyticDB for MySQL.

- ADDDATE: adds a time interval to the specified date.
- ADDTIME: adds a time interval to the specified time.
- CONVERT_TZ: converts from the time zone specified by from_tz to the time zone specified by to_tz, and returns the resulting value.
- CURDATE: returns the current date.
- CURTIME: returns the current time.
- DATE: returns the date part of a date or datetime expression.
- DATE_FORMAT: returns a date as a string in the specified format.
- SUBDATE/DATE_SUB: subtracts the specified interval from a date.
- DATEDIFF: subtracts a date from another.
- DAY/DAYOFMONTH: returns the day of the month for a date. Valid values: 1 to 31.
- DAYNAME: returns the day of the week for a date, such as Monday.
- DAYOFWEEK: returns the day of the week as a numeric value for a date.
- DAYOFYEAR: returns the day of the year for a date.
- EXTRACT: returns one or more separate parts of a date or time. For example, this function can return the year, month, day, hour, or minute of a date or time.
- FROM_DAYS: returns a DATE value based on the parameter N indicating the number of days.
- FROM_UNIXTIME: returns a UNIX timestamp in the specified format.
- HOUR: returns the hour part of a time.
- LAST_DAY: returns the last day of the month for a date or datetime.
- LOCALTIME/LOCALTIMESTAMP/NOW: returns the current timestamp.
- MAKEDATE: returns a date based on the year and dayofyear parameters.
- MAKETIME: returns a time based on the hour, minute, and second parameters.
- MINUTE: returns the minute part of a time.
- MONTH: returns the month of a date.
- MONTHNAME: returns the full name of the month for a date.
- PERIOD_ADD: adds a number of months specified by N to the period specified by P.
- PERIOD_DIFF: returns the number of months between two periods.

- QUARTER: returns the quarter of the year for a date.
- SEC_TO_TIME: converts a quantity of seconds to a time.
- SECOND: returns the second part of the specified time.
- STR_TO_DATE: converts a string to a date or datetime in the specified format.
- SUBTIME: subtracts a time interval from a date.
- SYSDATE: returns the system time.
- TIME: returns the time part of the date or datetime expression specified by expr as a string.
- TIME_FORMAT: returns a time as a string in the specified format.
- TIME_TO_SEC: converts a time to a quantity of seconds.
- TIMEDIFF: subtracts a time from another.
- TIMESTAMP: returns the date or datetime expression specified by expr as a date or datetime value.
- TIMESTAMPADD: adds an interval to a date or datetime expression.
- TIMESTAMPDIFF: subtracts an interval from a date or datetime expression.
- TO_DAYS: returns the number of days since year 0 based on the specified date.
- TO_SECONDS: returns the number of seconds that have elapsed since year 0 based on the specified time.
- UNIX_TIMESTAMP: returns the timestamp for the current time. The timestamp follows the UNIX time format. It is the number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970.
- UTC_DATE: returns the UTC date.
- UTC_TIME: returns the UTC time.
- UTC_TIMESTAMP: returns the UTC timestamp.
- WEEK: returns the week number for a date.
- WEEKDAY: returns the weekday for a date.
- WEEKOFYEAR: returns the calendar week for a date.
- YEAR: returns the year part of a date.
- YEARWEEK: returns the year and week of a date.

ADDDATE

```
ADDDATE(date,INTERVAL expr unit)
```

```
ADDDATE(expr,days
```

- Parameter types:

```
adddate(date, INTERVAL expr unit)
adddate(timestamp, INTERVAL expr unit)
adddate(datetime, INTERVAL expr unit)
adddate(varchar, INTERVAL expr unit)
adddate(date, varchar)
adddate(date, bigint)
adddate(datetime, bigint)
adddate(datetime, varchar)
adddate(timestamp, varchar)
adddate(timestamp, bigint)
adddate(varchar, bigint)
adddate(varchar, varchar)
```

- Return value type: DATE.
- Description: This function adds a time interval to the specified date.
  - Valid values of unit: second, minute, hour, day, month, year, minute_second, hour_second, hour_minute, day_second, day_minute, day_hour, and year_month. Default value of unit: day.
  - days and expr: This function returns the value of expr plus days.
- Example:

```
select adddate(date '2001-1-22',interval '3' day);
+--------------------------------------------+
| adddate(DATE '2001-1-22', INTERVAL  '3' DAY) |
+--------------------------------------------+
| 2001-01-25                                 |
```

```
select adddate(timestamp '2001-1-22',interval '3' day);
+-------------------------------------------------+
| adddate(TIMESTAMP '2001-1-22', INTERVAL  '3' DAY) |
+-------------------------------------------------+
| 2001-01-25 00:00:00                             |
```

```
select adddate(datetime '2001-1-22',interval '3' day);
+-------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' DAY) |
+-------------------------------------------------+
| 2001-01-25 00:00:00                             |
```

```
select adddate('2001-1-22',interval '3' day);
+---------------------------------------+
| adddate('2001-1-22', INTERVAL  '3' DAY) |
+---------------------------------------+
| 2001-01-25
```

```
select adddate(datetime '2001-1-22',interval '3' second);
+--------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' SECOND) |
+--------------------------------------------------+
```

```
|                          2001-01-22 00:00:03 |
```

```
select adddate(datetime '2001-1-22',interval '3' minute);
+-----------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' MINUTE) |
+-----------------------------------------------------+
|                          2001-01-22 00:03:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' hour);
+---------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' HOUR) |
+---------------------------------------------------+
|                          2001-01-22 03:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' month);
+----------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' MONTH) |
+----------------------------------------------------+
|                          2001-04-22 00:00:00 |
select adddate(datetime '2001-1-22',interval '3' year);
+---------------------------------------------------+
| adddate(DATETIME '2001-1-22', INTERVAL  '3' YEAR) |
+---------------------------------------------------+
|                          2004-01-22 00:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' hour_second) as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-22 03:00:00 |
select adddate(datetime '2001-1-22',interval '3' hour_minute) as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-22 03:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' day_second) as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-25 00:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' minute_second) as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-22 00:03:00 |
```

```
adddate(datetime '2001-1-22',interval '3' day_minute) as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-25 00:00:00 |
```

```
select adddate(datetime '2001-1-22',interval '3' day_hour) as result;
+---------------------+
```

```
| result              |
+---------------------+
| 2001-01-25 00:00:00 |
```

```
select adddate(datetime '2001-1-22 12:32:1',interval '4' year_month) as result;
+---------------------+
| result              |
+---------------------+
| 2005-01-22 12:32:01 |
```

```
select adddate('2001-1-22','3');
+--------------------------+
| adddate('2001-1-22', '3') |
+--------------------------+
| 2001-01-25               |
```

```
select adddate('2001-1-22',3);
+-------------------------+
| adddate('2001-1-22', 3) |
+-------------------------+
| 2001-01-25              |
```

```
select adddate(datetime '2001-1-22 12:12:32',3);
+-----------------------------------------+
| adddate(DATETIME '2001-1-22 12:12:32', 3) |
+-----------------------------------------+
|                   2001-01-25 12:12:32 |
```

```
select adddate(datetime '2001-1-22 12:12:32','3');
+-----------------------------------------+
| adddate(DATETIME '2001-1-22 12:12:32', '3') |
+-----------------------------------------+
|                    2001-01-25 12:12:32 |
```

```
select adddate(timestamp '2001-1-22 12:12:32','3');
+-----------------------------------------+
| adddate(TIMESTAMP '2001-1-22 12:12:32', '3') |
+-----------------------------------------+
|                    2001-01-25 12:12:32 |
```

```
select adddate(timestamp '2001-1-22 12:12:32',3);
+-----------------------------------------+
| adddate(TIMESTAMP '2001-1-22 12:12:32', 3) |
+-----------------------------------------+
|                   2001-01-25 12:12:32 |
```

```
select adddate('2001-1-22 12:12:32',3);
+---------------------------------+
| adddate('2001-1-22 12:12:32', 3) |
+---------------------------------+
| 2001-01-25 12:12:32             |
```

```
select adddate('2001-1-22 12:12:32','3');
+---------------------------------+
| adddate('2001-1-22 12:12:32', '3') |
+---------------------------------+
```

```
| 2001-01-25 12:12:32              |
```

ADDTIME

```
ADDTIME(expr1,expr2)
```

- Description: This function adds the time specified by `expr2` to the time specified by `expr1` and returns the result.
- Parameter types:

```
addtime(date,varchar)
addtime(time,varchar)
addtime(datetime,varchar)
addtime(timestamp,varchar)
addtime(varchar,varchar)
```

- Return value type: VARCHAR.
- Example:

```
select addtime(date '1998-01-01','01:01:01');
+--------------------------------------+
| addtime(DATE '1998-01-01', '01:01:01') |
+--------------------------------------+
| 1998-01-01 01:01:01                  |
```

```
select addtime(time '00:00:00','01:01:01');
+-------------------------------------+
| addtime(TIME '00:00:00', '01:01:01') |
+-------------------------------------+
| 01:01:01                            |
```

```
select addtime(datetime '2001-1-22 00:00:00','01:01:01');
+-------------------------------------------------+
| addtime(DATETIME '2001-1-22 00:00:00', '01:01:01') |
+-------------------------------------------------+
| 2001-01-22 01:01:01                             |
```

```
select addtime(timestamp '2001-1-22 00:00:00','01:01:01');
+-------------------------------------------------+
| addtime(TIMESTAMP '2001-1-22 00:00:00', '01:01:01') |
+-------------------------------------------------+
| 2001-01-22 01:01:01                             |
```

```
select addtime('2001-1-22 00:00:00','01:01:01');
+-----------------------------------------+
| addtime('2001-1-22 00:00:00', '01:01:01') |
+-----------------------------------------+
```

```
| 2001-01-22 01:01:01                  |
```

CONVERT_TZ

```
CONVERT_TZ(dt,from_tz,to_tz)
```

- Description: This function converts a datetime value dt from the time zone specified by from_tz to the time zone specified by to_tz and returns the result.
- Parameter types:

```
convert_tz(varchar, varchar, varchar)
```

- Return value type: DATETIME.
- Example:

```
select convert_tz('2004-01-01 12:00:00','+00:00','+10:00');
+-----------------------------------------------------+
| convert_tz('2004-01-01 12:00:00', '+00:00', '+10:00') |
+-----------------------------------------------------+
|                              2004-01-01 22:00:00 |
```

```
select convert_tz('2004-01-01 12:00:00','GMT','MET');
+-----------------------------------------------+
| convert_tz('2004-01-01 12:00:00', 'GMT', 'MET') |
+-----------------------------------------------+
|                          2004-01-01 13:00:00 |
```

CURDATE

```
CURDATE()
```

- Description: This function returns the current date.
- Return value type: DATE.
- Example:

```
select curdate;
+------------+
| curdate()  |
+------------+
| 2019-05-25 |
```

CURTIME

```
CURTIME()
```

- Description: This function returns the current time.
- Return value type: TIME.
- Example:

```
select curtime();
```

```
+--------------+
| curtime()    |
+--------------+
| 14:39:22.109 |
```

DATE

```
DATE(expr)
```

- Description: This function returns the date part of a date or datetime expression.
- Parameter types:

```
date(timestamp)
date(datetime)
date(varchar)
```

- Return value type: DATE.
- Example:

```
select date(timestamp '2003-12-31 01:02:03');
+--------------------------------------+
| date(TIMESTAMP '2003-12-31 01:02:03') |
+--------------------------------------+
| 2003-12-31                           |
```

```
select date(datetime '2003-12-31 01:02:03');
+--------------------------------------+
| date(DATETIME '2003-12-31 01:02:03') |
+--------------------------------------+
| 2003-12-31                           |
```

```
select date('2003-12-31 01:02:03');
+----------------------------+
| date('2003-12-31 01:02:03') |
+----------------------------+
| 2003-12-31                 |
```

DATE_FORMAT

```
DATE_FORMAT(date,format)
```

- Description: This function returns a date as a string in the specified format. The following table describes the format specifiers.

%a	The abbreviated day of a week. Valid values: Sun to Sat.
%b	The abbreviated month name. Valid values : Jan to Dec.
%c	The month in the numeric format. Valid values: 0 to 12.

%d	The day of the month in the numeric format. Valid values: 00 to 31.
%e	The day of the month in the numeric format. Valid values: 0 to 31.
%f	The microseconds. Valid values: 000000 to 999999.
%H	The hour. Valid values: 00 to 23.
%h	The hour. Valid values: 01 to 12.
%I	The hour. Valid values: 01 to 12.
%i	The minutes in the numeric format. Valid values: 00 to 59.
%j	The day of the year. Valid values: 001 to 366.
%k	The hour. Valid values: 0 to 23.
%l	The hour. Valid values: 1 to 12.
%M	The name of the month. Valid values: January to December.
%m	The month in the numeric format. Valid values: 00 to 12.
%p	The abbreviated time period in the 12-hour format. Valid values: AM and PM.
%r	The time in the 12-hour format (hh:mm:ss AM or hh:mm:ss PM).
%S	The seconds. Valid values: 00 to 59.
%s	The seconds. Valid values: 00 to 59.
%T	The time in the 24-hour format (hh:mm:ss).
%v	The number of the week in the year. Monday is the first day of a week. This specifier applies to WEEK() mode 3 and is used with %x.
%W	The name of the weekday. Valid values: Sunday to Saturday.

%x	The year of the week in the numeric format . Monday is the first day of a week. This specifier is used with %v, and the value contains four digits.
%Y	The year in the four-digit format.
%y	The year in the two-digit format.
%%	The percent sign (%).
%x	x, for any "x" not listed above.

- Parameter types:

```
date_format(timestamp, varchar)
date_format(varchar, varchar)
date_format(datetime, varchar)
date_format(date, varchar)
```

- Return value type: VARCHAR.
- Example:

```
select date_format(timestamp '2019-05-27 13:23:00', '%W %M %Y')as result;
+-----------------+
| result          |
+-----------------+
| Monday May 2019 |
```

```
select date_format(timestamp '2019-05-27 13:23:00', '%W %M %Y')as result;
+-----------------+
| result          |
+-----------------+
| Monday May 2019 |
```

```
select date_format(timestamp '2019-05-27 13:23:00', '%W %M %Y')as result;
+-----------------+
| result          |
+-----------------+
| Monday May 2019 |
```

```
select date_format(date '2019-05-27', '%W %M %Y')as result;
+-----------------+
| result          |
+-----------------+
```

```
| Monday May 2019 |
```

SUBDATE/DATE_SUB

```
DATE_SUB(date,INTERVAL expr unit)
```

- Description: This function subtracts the interval specified by INTERVAL from the date specified by date.

  Valid values of unit: second, minute, hour, day, month, year, minute_second, hour_second, hour_minute, day_second, day_minute, day_hour, and year_month. Default value of unit: day.

- Parameter types:

```
subdate(date, INTERVAL expr unit)
subdate(timestamp, INTERVAL expr unit)
subdate(datetime, INTERVAL expr unit)
subdate(varchar, INTERVAL expr unit)
subdate(date, bigint)
subdate(date, varchar)
subdate(datetime, bigint)
subdate(datetime, varchar)
subdate(timestamp, bigint)
subdate(timestamp, varchar)
subdate(varchar, bigint)
subdate(varchar, varchar)
```

- Return value type: DATE.
- Example:

```
select date_sub(date '2001-1-22',interval '3' day);
+--------------------------------------------+
| date_sub(DATE '2001-1-22', INTERVAL  '3' DAY) |
+--------------------------------------------+
| 2001-01-19                                 |
```

```
select date_sub(timestamp '2001-1-22 00:00:00',interval '3' day)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-19 00:00:00 |
```

```
select date_sub(datetime '2001-1-22 00:00:00',interval '3' day)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-19 00:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' day);
+-------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' DAY) |
+-------------------------------------------------+
```

```
| 2001-01-19 00:00:00                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' second);
+-----------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' SECOND) |
+-----------------------------------------------------+
| 2001-01-21 23:59:57                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' minute);
+-----------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' MINUTE) |
+-----------------------------------------------------+
| 2001-01-21 23:57:00                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' hour);
+---------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' HOUR) |
+---------------------------------------------------+
| 2001-01-21 21:00:00                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' month);
+----------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' MONTH) |
+----------------------------------------------------+
| 2000-10-22 00:00:00                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' year);
+---------------------------------------------------+
| date_sub('2001-1-22 00:00:00', INTERVAL  '3' YEAR) |
+---------------------------------------------------+
| 1998-01-22 00:00:00                          |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' minute_second)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-21 23:57:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' hour_second)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-21 21:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' hour_minute)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-21 21:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' day_second)as result;
+---------------------+
| result              |
+---------------------+
```

```
| 2001-01-19 00:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' day_minute)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-19 00:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' day_hour)as result;
+---------------------+
| result              |
+---------------------+
| 2001-01-19 00:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',interval '3' year_month)as result;
+---------------------+
| result              |
+---------------------+
| 1998-01-22 00:00:00 |
```

```
select date_sub(date '2001-1-22 00:00:00',3);
+---------------------------------------+
| date_sub(DATE '2001-1-22 00:00:00', 3) |
+---------------------------------------+
| 2001-01-19                            |
```

```
select date_sub(date '2001-1-22 00:00:00','3');
+-----------------------------------------+
| date_sub(DATE '2001-1-22 00:00:00', '3') |
+-----------------------------------------+
| 2001-01-19                              |
```

```
select date_sub(datetime '2001-1-22 00:00:00',3);
+-------------------------------------------+
| date_sub(DATETIME '2001-1-22 00:00:00', 3) |
+-------------------------------------------+
|                   2001-01-19 00:00:00 |
```

```
select date_sub(datetime '2001-1-22 00:00:00','3');
+---------------------------------------------+
| date_sub(DATETIME '2001-1-22 00:00:00', '3') |
+---------------------------------------------+
|                    2001-01-19 00:00:00 |
```

```
select date_sub(timestamp '2001-1-22 00:00:00',3);
+--------------------------------------------+
| date_sub(TIMESTAMP '2001-1-22 00:00:00', 3) |
+--------------------------------------------+
|                   2001-01-19 00:00:00 |
```

```
select date_sub(timestamp '2001-1-22 00:00:00','3');
+----------------------------------------------+
| date_sub(TIMESTAMP '2001-1-22 00:00:00', '3') |
+----------------------------------------------+
|                    2001-01-19 00:00:00 |
```

```
select date_sub('2001-1-22 00:00:00',3);
+----------------------------------+
| date_sub('2001-1-22 00:00:00', 3) |
+----------------------------------+
| 2001-01-19 00:00:00              |
```

```
select date_sub('2001-1-22 00:00:00','3');
+-----------------------------------+
| date_sub('2001-1-22 00:00:00', '3') |
+-----------------------------------+
| 2001-01-19 00:00:00               |
```

DATEDIFF

```
DATEDIFF(expr1,expr2)
```

- Description: This function subtracts the date specified by `expr2` from the date specified by `expr1`.
- Parameter types:

```
datediff(varchar, varchar)
datediff(datetime, varchar)
datediff(varchar, datetime)
datediff(datetime, datetime)
datediff(varchar, timestamp)
datediff(timestamp, timestamp)
datediff(timestamp, varchar)
datediff(date, date)
datediff(date, varchar)
datediff(varchar, date)
```

- Return value type: BIGINT.
- Example:

```
select datediff('2007-12-31 23:59:59','2007-12-30');
+-----------------------------------------------+
| datediff('2007-12-31 23:59:59', '2007-12-30') |
+-----------------------------------------------+
|                                             1 |
```

```
select datediff(datetime '2007-12-31 23:59:59','2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff('2007-12-31 23:59:59',datetime '2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff(datetime '2007-12-31 23:59:59',datetime '2007-12-30')as result;
```

```
+--------+
| result |
+--------+
|      1 |
```

```
select datediff('2007-12-31 23:59:59',timestamp '2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff(timestamp '2007-12-31 23:59:59',timestamp '2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff(timestamp '2007-12-31 23:59:59','2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff(date '2007-12-31 23:59:59',date '2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff(date '2007-12-31 23:59:59','2007-12-30')as result;
+--------+
| result |
+--------+
|      1 |
```

```
select datediff('2008-12-31',date '2007-12-30');
+-----------------------------------------+
| datediff('2008-12-31', DATE '2007-12-30') |
+-----------------------------------------+
                                      367
```

DAY/DAYOFMONTH

```
DAY(date)
DAYOFMONTH(date)
```

- Description: This function returns the day of the month for the date specified by `date`.

  Valid values: [1,31].

- Parameter types:

```
dayofmonth(timestamp)
dayofmonth(datetime)
dayofmonth(date)
dayofmonth(time)
```

```
dayofmonth(varchar)
```

- Return value type: BIGINT.
- Example:

```
select dayofmonth(timestamp '2007-02-03 12:23:09');
+-------------------------------------------+
| dayofmonth(TIMESTAMP '2007-02-03 12:23:09') |
+-------------------------------------------+
|                                         3 |
```

```
select dayofmonth(date '2007-02-03');
+-----------------------------+
| dayofmonth(DATE '2007-02-03') |
+-----------------------------+
|                           3 |
```

```
select dayofmonth(time '17:01:10');
+-----------------------------+
| dayofmonth(TIME '17:01:10') |
+-----------------------------+
|                          30 |
```

```
select day('2007-02-03');
+-------------------+
| day('2007-02-03') |
+-------------------+
|                 3 |
```

```
select dayofmonth(datetime '2007-02-03 00:00:00');
+-----------------------------------------+
| dayofmonth(DATETIME '2007-02-03 00:00:00') |
+-----------------------------------------+
|                                       3 |
```

DAYNAME

```
DAYNAME(date)
```

- Description: This function returns the day of the week for a date, such as `Monday`.
- Parameter types:

```
dayname(timestamp)
dayname(datetime)
dayname(date)
dayname(varchar)
```

- Return value type: VARCHAR.
- Example:

```
select dayname(timestamp '2007-02-03 00:00:00');
+-----------------------------------------+
| dayname(TIMESTAMP '2007-02-03 00:00:00') |
+-----------------------------------------+
```

```
| Saturday                              |
```

```
select dayname(datetime '2007-02-03 00:00:00');
+---------------------------------------+
| dayname(DATETIME '2007-02-03 00:00:00') |
+---------------------------------------+
| Saturday                              |
```

```
select dayname(date '2007-02-04');
+---------------------------+
| dayname(DATE '2007-02-04') |
+---------------------------+
| Sunday                    |
```

```
select dayname('2007-02-03');
+----------------------+
| dayname('2007-02-03') |
+----------------------+
| Saturday             |
```

DAYOFWEEK

```
DAYOFWEEK(date)
```

- Description: This function returns the day of the week as a numeric value for a date, where `1` is for Sunday, `2` for Monday, and `7` for Saturday.
- Parameter types:

```
dayofweek(timestamp)
dayofweek(datetime)
dayofweek(date)
dayofweek(varchar)
```

- Return value type: BIGINT.
- Example:

```
select dayofweek(timestamp '2007-02-03 00:00:00');
+-------------------------------------------+
| dayofweek(TIMESTAMP '2007-02-03 00:00:00') |
+-------------------------------------------+
|                                         7 |
```

```
select dayofweek(datetime '2007-02-03 00:00:00');
+-----------------------------------------+
| dayofweek(DATETIME '2007-02-03 00:00:00') |
+-----------------------------------------+
|                                       7 |
```

```
select dayofweek(date '2007-02-03');
+-----------------------------+
| dayofweek(DATE '2007-02-03') |
+-----------------------------+
|                           7 |
```

```
select dayofweek('2007-02-03');
```

```
+------------------------+
| dayofweek('2007-02-03') |
+------------------------+
|         7              |
```

DAYOFYEAR

```
DAYOFYEAR(date)
```

- Description: This function returns the day of the year for a date. Valid values: [1,366].
- Parameter types:

```
dayofyear(timestamp)
dayofyear(datetime)
dayofyear(date)
dayofyear(varchar)
```

- Return value type: BIGINT.
- Example:

```
select dayofyear(timestamp '2007-02-03 00:12:12');
+-----------------------------------------+
| dayofyear(TIMESTAMP '2007-02-03 00:12:12') |
+-----------------------------------------+
|                                      34 |
```

```
select dayofyear(datetime '2007-02-03 00:12:12');
+-----------------------------------------+
| dayofyear(DATETIME '2007-02-03 00:12:12') |
+-----------------------------------------+
|                                      34 |
```

```
select dayofyear(date '2007-02-03');
+-----------------------------+
| dayofyear(DATE '2007-02-03') |
+-----------------------------+
|                          34 |
```

```
select dayofyear('2007-02-03');
+-------------------------+
| dayofyear('2007-02-03') |
+-------------------------+
```

```
|        34        |
```

EXTRACT

```
EXTRACT(unit FROM date)
```

- Description: This function returns one or more separate parts of a date or time in the specified unit. For example, this function can return the year, month, day, hour, or minute of a date or time.

  Valid values of unit: second, minute, hour, day, month, year, minute_second, hour_second, hour_minute, day_second, day_minute, day_hour, and year_month.
- Supported input parameter types: VARCHAR, TIMESTAMP, DATETIME, and TIME.
- Return value type: BIGINT.
- Example:

```
select extract(second from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|    34 |
```

```
select extract(minute from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|    12 |
```

```
select extract(hour from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|     0 |
```

```
select extract(month from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|     7 |
```

```
select extract(minute_second from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|  1234 |
```

```
select extract(hour_second from '2019-07-02 12:12:34');
+--------+
| _col0  |
+--------+
```

```
| 121234 |
```

```
select extract(hour_minute from '2019-07-02 12:12:34');
+-------+
| _col0 |
+-------+
|  1212 |
```

```
select extract(day_second from '2019-07-02 12:12:34');
+---------+
| _col0   |
+---------+
| 2121234 |
```

```
select extract(day_hour from '2019-07-02 12:12:34');
+-------+
| _col0 |
+-------+
|   212 |
```

```
select extract(day from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
|     2 |
```

```
select extract(year_month from '2019-07-02 00:12:34');
+--------+
| _col0  |
+--------+
| 201907 |
```

```
select extract(day_minute from '2019-07-02 00:12:34');
+-------+
| _col0 |
+-------+
| 20012 |
```

```
select extract(year from timestamp  '2019-05-30');
+-------+
| _col0 |
+-------+
|  2019 |
```

```
select extract(year from datetime  '2019-05-30');
+-------+
| _col0 |
+-------+
|  2019 |
```

```
select extract(year from time '15:23:22');
+-------+
| _col0 |
+-------+
```

```
| 2019 |
```

FROM_DAYS

```
FROM_DAYS(N)
```

- Description: This function returns a DATE value based on the parameter N indicating the number of days.
- Parameter types:

```
from_days(varchar)
from_days(bigint)
```

- Return value type: DATE.
- Example:

```
select from_days(730669);
+------------------+
| from_days(730669) |
+------------------+
| 2000-07-03       |
```

```
select from_days('730669');
+--------------------+
| from_days('730669') |
+--------------------+
| 2000-07-03         |
```

FROM_UNIXTIME

```
FROM_UNIXTIME(unix_timestamp[,format])
```

- Description: This function returns the UNIX timestamp specified by unixtime in the specified format.

  The format parameter conforms to the format in the DATE_FORMAT function.
- Parameter types:

```
from_unixtime(varchar, varchar)
from_unixtime(varchar)
from_unixtime(double, varchar)
from_unixtime(double)
```

- Return value type: DATETIME.
- Example:

```
select from_unixtime('1447430881','%Y %M %h:%i:%s %x');
+------------------------------------------------+
| from_unixtime('1447430881', '%Y %M %h:%i:%s %x') |
+------------------------------------------------+
```

```
| 2015 November 12:08:01 2015                    |
```

```
select from_unixtime('1447430881');
+----------------------------+
| from_unixtime('1447430881') |
+----------------------------+
|        2015-11-14 00:08:01 |
```

```
select from_unixtime(1447430881);
+--------------------------+
| from_unixtime(1447430881) |
+--------------------------+
|      2015-11-14 00:08:01 |
```

```
select from_unixtime(1447430881,'%Y %M %h:%i:%s %x');
+-----------------------------------------------+
| from_unixtime(1447430881, '%Y %M %h:%i:%s %x') |
+-----------------------------------------------+
| 2015 November 12:08:01 2015                    |
```

HOUR

```
HOUR(time)
```

- Description: This function returns the hour part of a time.
- Parameter types:

```
hour(timestamp)
hour(datetime)
hour(date)
hour(time)
hour(varchar)
```

- Return value type: BIGINT.
- Example:

```
select hour(timestamp '2019-12-07 10:05:03');
+--------------------------------------+
| hour(TIMESTAMP '2019-12-07 10:05:03') |
+--------------------------------------+
|                                   10 |
```

```
select hour(datetime '2019-12-07 10:05:03');
+--------------------------------------+
| hour(DATETIME '2019-12-07 10:05:03') |
+--------------------------------------+
|                                   10 |
```

```
select hour(date '2019-12-07');
+-----------------------+
| hour(DATE '2019-12-07') |
+-----------------------+
|                     0 |
```

```
select hour(time '10:05:03');
```

```
+----------------------+
| hour(TIME '10:05:03') |
+----------------------+
|                   10 |
```

```
select hour('10:05:03');
+-----------------+
| hour('10:05:03') |
+-----------------+
|      10         |
```

LAST_DAY

```
LAST_DAY(date)
```

- Description: This function returns the last day of the month for a date or datetime.
- Parameter types:

```
last_day(varchar)
last_day(timestamp)
last_day(datetime)
last_day(date)
```

- Return value type: DATE.
- Example:

```
select last_day('2003-02-05');
+-----------------------+
| last_day('2003-02-05') |
+-----------------------+
| 2003-02-28            |
```

```
select last_day(timestamp '2003-02-05 12:12:12');
+-----------------------------------------+
| last_day(TIMESTAMP '2003-02-05 12:12:12') |
+-----------------------------------------+
| 2003-02-28                              |
```

```
select last_day(datetime '2003-02-05 12:12:12');
+-----------------------------------------+
| last_day(DATETIME '2003-02-05 12:12:12') |
+-----------------------------------------+
| 2003-02-28                              |
```

```
select last_day(date '2003-02-05');
+----------------------------+
| last_day(DATE '2003-02-05') |
+----------------------------+
| 2003-02-28                 |
```

LOCALTIME/LOCALTIMESTAMP/NOW

```
localtime
localtime()
localtimestamp
localtimestamp()
```

```
now()
```

- Description: This function returns the current timestamp.
- Return value type: DATETIME.
- Example:

```
select now();
+---------------------+
| now()               |
+---------------------+
| 2019-05-25 00:28:37
```

```
select localtime;
+---------------------+
| localtime()         |
+---------------------+
| 2019-05-28 20:44:25 |
```

```
select localtime();
+---------------------+
| localtime()         |
+---------------------+
| 2019-05-31 17:37:36 |
```

```
select localtimestamp;
+---------------------+
| localtimestamp()    |
+---------------------+
| 2019-05-28 20:44:44 |
```

```
select localtimestamp();
+---------------------+
| localtimestamp()    |
+---------------------+
| 2019-05-31 17:38:13 |
```

MAKEDATE

```
MAKEDATE(year,dayofyear)
```

- Description: This function returns a date based on the `year` and `dayofyear` parameters.
- Parameter types:

```
makedate(bigint, bigint)
makedate(varchar, varchar)
```

- Return value type: DATE.
- Example:

```
select makedate(2011,31), makedate(2011,32);
+-------------------+-------------------+
| makedate(2011, 31) | makedate(2011, 32) |
+-------------------+-------------------+
```

```
| 2011-01-31        | 2011-02-01        |
```

```
select makedate('2011','31'), makedate('2011','32');
+-----------------------+-----------------------+
| makedate('2011', '31') | makedate('2011', '32') |
+-----------------------+-----------------------+
| 2011-01-31            | 2011-02-01            |
```

MAKETIME

```
MAKETIME(hour,minute,second)
```

- Description: This function returns a time based on the `hour`, `minute`, and `second` parameters.
- Parameter types:

```
maketime(bigint, bigint, bigint)
maketime(varchar, varchar, varchar)
```

- Return value type: TIME.
- Example:

```
select maketime(12,15,30);
+---------------------+
| maketime(12, 15, 30) |
+---------------------+
| 12:15:30            |
```

```
select maketime('12','15','30');
+---------------------------+
| maketime('12', '15', '30') |
+---------------------------+
| 12:15:30                  |
```

MINUTE

```
MINUTE(time)
```

- Description: This function returns the minute part of a time.
- Parameter types:

```
minute(timestamp)
minute(datetime)
minute(date)
minute(time)
minute(varchar)
```

- Return value type: BIGINT.
- Example:

```
select minute(timestamp '2008-02-03 10:05:03');
+----------------------------------------+
| minute(TIMESTAMP '2008-02-03 10:05:03') |
```

```
+----------------------------------------+
|                                      5 |
```

```
select minute(datetime '2008-02-03 10:05:03');
+----------------------------------------+
| minute(DATETIME '2008-02-03 10:05:03') |
+----------------------------------------+
|                                      5 |
```

```
select minute(date '2008-02-03');
+--------------------------+
| minute(DATE '2008-02-03') |
+--------------------------+
|                        0 |
```

```
select minute(time '12:12:12');
+-------------------------+
| minute(TIME '12:12:12') |
+-------------------------+
|                      12 |
```

```
select minute('2008-02-03 10:05:03');
+-------------------------------+
| minute('2008-02-03 10:05:03') |
+-------------------------------+
|          5                    |
```

MONTH

```
MONTH(date)
```

- Description: This function returns the month part of a date.
- Parameter types:

```
month(timestamp)
month(datetime)
month(date)
month(time)
month(varchar)
```

- Return value type: BIGINT.
- Example:

```
select month(timestamp '2008-02-03 00:00:00');
+----------------------------------------+
| month(TIMESTAMP '2008-02-03 00:00:00') |
+----------------------------------------+
|                                      2 |
```

```
select month(datetime '2008-02-03 00:00:00');
+---------------------------------------+
| month(DATETIME '2008-02-03 00:00:00') |
+---------------------------------------+
```

```
|                            2 |
```

```
select month(date '2008-02-03');
+-------------------------+
| month(DATE '2008-02-03') |
+-------------------------+
|                       2 |
```

The MONTH function can also return the month when an SQL statement is executed. In the following example, 5 is returned for an SQL statement that is executed in May, 2019.

```
select month(time '12:12:12');
+-----------------------+
| month(TIME '12:12:12') |
+-----------------------+
|                     5 |
```

```
select month('2008-02-03');
+--------------------+
| month('2008-02-03') |
+--------------------+
|                  2 |
```

MONTHNAME

```
MONTHNAME(date)
```

- Description: This function returns the full name of the month for a date.
- Parameter types:

```
monthname(timestamp)
monthname(datetime)
monthname(date)
monthname(varchar)
```

- Return value type: VARCHAR.
- Example:

```
select monthname(timestamp '2008-02-03 00:00:00');
+-------------------------------------------+
| monthname(TIMESTAMP '2008-02-03 00:00:00') |
+-------------------------------------------+
| February                                  |
```

```
select monthname(datetime '2008-02-03 00:00:00');
+-------------------------------------------+
| monthname(DATETIME '2008-02-03 00:00:00') |
+-------------------------------------------+
| February                                  |
```

```
select monthname(date '2008-02-03');
+----------------------------+
| monthname(DATE '2008-02-03') |
+----------------------------+
```

```
| February                  |
```

```
select monthname('2008-02-03');
+------------------------+
| monthname('2008-02-03') |
+------------------------+
| February               |
```

PERIOD_ADD

```
PERIOD_ADD(P,N)
```

- Description: This function adds a number of months specified by `N` to the period specified by `P`.
- Parameter types:

```
period_add(bigint, bigint)
period_add(varchar, varchar)
period_add(varchar, bigint)
```

- Return value type: BIGINT.
- Example:

```
select period_add(200801,2);
+----------------------+
| period_add(200801, 2) |
+----------------------+
|               200803 |
```

```
select period_add('200801','2');
+---------------------------+
| period_add('200801', '2') |
+---------------------------+
|                  200803 |
```

```
select period_add('200801',2);
+------------------------+
| period_add('200801', 2) |
+------------------------+
|               200803 |
```

PERIOD_DIFF

```
PERIOD_DIFF(P1,P2)
```

- Description: This function returns the number of months between the two periods specified by `P1` and `P2`.
- Parameter types:

```
period_diff(bigint, bigint)
```

```
period_diff(varchar, varchar)
```

- Return value type: BIGINT.
- Example:

```
select period_diff(200802,200703);
+----------------------------+
| period_diff(200802, 200703) |
+----------------------------+
|                         11 |
```

```
select period_diff('200802','200703');
+--------------------------------+
| period_diff('200802', '200703') |
+--------------------------------+
|                             11 |
```

QUARTER

```
QUARTER(date)
```

- Description: This function returns the quarter of the year for a date. Valid values: [1,4].
- Parameter types:

```
quarter(datetime)
quarter(varchar)
quarter(timestamp)
quarter(date)
```

- Return value type: BIGINT.
- Example:

```
select quarter(datetime '2008-04-01 12:12:12');
+----------------------------------------+
| quarter(DATETIME '2008-04-01 12:12:12') |
+----------------------------------------+
|                                      2 |
```

```
select quarter('2008-04-01');
+----------------------+
| quarter('2008-04-01') |
+----------------------+
|                    2 |
```

```
select quarter(timestamp '2008-04-01 12:12:12');
+-----------------------------------------+
| quarter(TIMESTAMP '2008-04-01 12:12:12') |
+-----------------------------------------+
|                                       2 |
```

```
select quarter(date '2008-04-01');
+---------------------------+
| quarter(DATE '2008-04-01') |
+---------------------------+
```

```
|                    2 |
```

SEC_TO_TIME

```
SEC_TO_TIME(seconds)
```

- Description: This function converts a quantity of seconds specified by `seconds` to a time.
- Parameter types:

```
sec_to_time(bigint)
sec_to_time(varchar)
```

- Return value type: TIME.
- Example:

```
select sec_to_time(2378);
+------------------+
| sec_to_time(2378) |
+------------------+
| 00:39:38         |
```

```
select sec_to_time('2378');
+--------------------+
| sec_to_time('2378') |
+--------------------+
| 00:39:38
```

SECOND

```
SECOND(time)
```

- Description: This function returns the second part of the specified time. Valid values: [0, 59].
- Parameter types:

```
second(timestamp)
second(datetime)
second(date)
second(time)
second(varchar)
```

- Return value type: BIGINT.
- Example:

```
select second(timestamp '2019-03-12 12:13:14');
+----------------------------------------+
| second(TIMESTAMP '2019-03-12 12:13:14') |
+----------------------------------------+
|                                     14 |
```

```
select second(datetime '2019-03-12 12:13:14');
+----------------------------------------+
| second(DATETIME '2019-03-12 12:13:14') |
```

```
+----------------------------------------+
|                                     14 |
```

```
select second(date '2019-03-12');
+--------------------------+
| second(DATE '2019-03-12') |
+--------------------------+
|                        0 |
```

```
select second(time '12:13:14');
+-------------------------+
| second(TIME '12:13:14') |
+-------------------------+
|                      14 |
```

```
select second('12:12:23');
+--------------------+
| second('12:12:23') |
+--------------------+
|                 23 |
```

STR_TO_DATE

```
STR_TO_DATE(str,format)
```

- Description: This function converts a string to a date or datetime in the specified format.

  The format parameter conforms to the format in the DATE_FORMAT function.
- Parameter types:

  ```
  str_to_date(varchar, varchar)
  ```
- Return value type: DATETIME.
- Example:

  ```
  select str_to_date('2017-01-06 10:20:30','%Y-%m-%d %H:%i:%s') as result;
  +---------------------+
  | result              |
  +---------------------+
  | 2017-01-06 10:20:30 |
  ```

SUBTIME

```
SUBTIME(expr1,expr2)
```

- Description: This function subtracts the interval specified by expr2 from the time specified by expr1.
- Parameter types:

  ```
  subtime(date, varchar)
  subtime(datetime, varchar)
  subtime(timestamp, varchar)
  subtime(time, varchar)
  ```

```
subtime(varchar, varchar)
```

- Return value type: DATETIME.
- Example:

```
select subtime(date '2018-10-31','0:1:1');
+-----------------------------------+
| subtime(DATE '2018-10-31', '0:1:1') |
+-----------------------------------+
|              2018-10-30 23:58:59 |
```

```
select subtime(datetime '2018-10-31 12:12:12','0:1:1');
+-------------------------------------------------+
| subtime(DATETIME '2018-10-31 12:12:12', '0:1:1') |
+-------------------------------------------------+
|                       2018-10-31 12:11:11 |
```

```
select subtime(timestamp '2018-10-31 12:12:12','0:1:1');
+-------------------------------------------------+
| subtime(TIMESTAMP '2018-10-31 12:12:12', '0:1:1') |
+-------------------------------------------------+
|                       2018-10-31 12:11:11 |
```

```
select subtime(time '12:12:12','0:1:1');
+---------------------------------+
| subtime(TIME '12:12:12', '0:1:1') |
+---------------------------------+
| 12:11:11                        |
+---------------------------------+
```

```
select subtime('2018-10-31 23:59:59','0:1:1');
+---------------------------------------+
| subtime('2018-10-31 23:59:59', '0:1:1') |
+---------------------------------------+
| 2018-10-31 23:58:58                   |
```

SYSDATE

```
SYSDATE()
```

- Description: This function returns the system time.
- Return value type: DATETIME.
- Example:

```
select sysdate();
+---------------------+
| sysdate()           |
+---------------------+
```

```
| 2019-05-26 00:47:21 |
```

TIME

```
TIME(expr)
```

- Description: This function returns the time part of the date or datetime expression specified by `expr` as a string.
- Parameter types:

```
time(varchar)
time(datetime)
time(timestamp)
```

- Return value type: VARCHAR.
- Example:

```
select time('2003-12-31 01:02:03');
+----------------------------+
| time('2003-12-31 01:02:03') |
+----------------------------+
| 01:02:03                   |
```

```
select time(datetime '2003-12-31 01:02:03');
+-------------------------------------+
| time(DATETIME '2003-12-31 01:02:03') |
+-------------------------------------+
| 01:02:03                            |
```

```
select time(timestamp '2003-12-31 01:02:03');
+--------------------------------------+
| time(TIMESTAMP '2003-12-31 01:02:03') |
+--------------------------------------+
| 01:02:03                             |
```

TIME_FORMAT

```
TIME_FORMAT(time,format)
```

- Description: This function returns the time specified by `time` as a string in the specified `format`.

  The `format` parameter conforms to the `format` in the DATE_FORMAT function.
- Parameter types:

```
time_format(varchar, varchar)
time_format(timestamp, varchar)
time_format(datetime, varchar)
time_format(time, varchar)
time_format(date, varchar)
```

- Return value type: VARCHAR.

- Example:

```
select time_format('12:00:00', '%H %k %h %I %l');
+-----------------------------------------+
| time_format('12:00:00', '%H %k %h %I %l') |
+-----------------------------------------+
| 12 12 12 12 12                          |
```

```
select time_format(timestamp '1998-01-01 23:00:00','%H %k %h %I %l')as result;
+----------------+
| result         |
+----------------+
| 23 23 11 11 11 |
```

```
select time_format(datetime '1998-01-01 23:00:00','%H %k %h %I %l')as result;
+----------------+
| result         |
+----------------+
| 23 23 11 11 11 |
```

```
select time_format(time '23:00:00','%H %k %h %I %l');
+-----------------------------------------------+
| time_format(TIME '23:00:00', '%H %k %h %I %l') |
+-----------------------------------------------+
| 23 23 11 11 11                                |
```

```
select time_format(date '1998-01-01','%H %k %h %I %l');
+-----------------------------------------------+
| time_format(DATE '1998-01-01', '%H %k %h %I %l') |
+-----------------------------------------------+
| 00 0 12 12 12                                 |
```

TIME_TO_SEC

```
TIME_TO_SEC(time)
```

- Description: This function converts the time specified by time to a quantity of seconds.
- Parameter types:

```
time_to_sec(varchar)
time_to_sec(datetime)
time_to_sec(timestamp)
time_to_sec(date)
time_to_sec(time)
```

- Return value type: BIGINT.
- Example:

```
select time_to_sec(datetime '2009-12-12 22:23:00');
+-------------------------------------------+
| time_to_sec(DATETIME '2009-12-12 22:23:00') |
+-------------------------------------------+
|                                     80580 |
```

```
select time_to_sec(timestamp '2009-12-12 22:23:00');
```

```
+-------------------------------------------+
| time_to_sec(TIMESTAMP '2009-12-12 22:23:00') |
+-------------------------------------------+
|                                     80580 |
```

```
select time_to_sec(date '2009-12-12');
+-------------------------------+
| time_to_sec(DATE '2009-12-12') |
+-------------------------------+
|                             0 |
```

```
select time_to_sec(time '12:12:12');
+-----------------------------+
| time_to_sec(TIME '12:12:12') |
+-----------------------------+
|                       43932 |
```

```
select time_to_sec('22:23:00');
+-------------------------+
| time_to_sec('22:23:00') |
+-------------------------+
|                   80580 |
```

TIMEDIFF

```
TIMEDIFF(expr1,expr2)
```

- Description: This function subtracts the time specified by `expr2` from the time specified by `expr1`. This function is equivalent to the SUBTIME function.
- Parameter types:

```
timediff(time, varchar)
timediff(time, time)
timediff(varchar, varchar)
```

- Return value type: DATETIME.
- Example:

```
select timediff(time '12:00:00','10:00:00');
+---------------------------------------+
| timediff(TIME '12:00:00', '10:00:00') |
+---------------------------------------+
| 02:00:00                              |
```

```
select timediff('12:00:00','10:00:00');
+---------------------------------+
| timediff('12:00:00', '10:00:00') |
+---------------------------------+
| 02:00:00                        |
```

```
select timediff(time '12:00:00',time '10:00:00');
+-------------------------------------------+
| timediff(TIME '12:00:00', TIME '10:00:00') |
+-------------------------------------------+
```

```
| 02:00:00                        |
```

TIMESTAMP

```
TIMESTAMP(expr)
```

- Description: This function returns the date or datetime expression specified by expr as a date or datetime value.
- Parameter types:

```
timestamp(date)
timestamp(varchar)
```

- Return value type: DATETIME.
- Example:

```
select timestamp(date '2019-05-27');
+-----------------------------+
| timestamp(DATE '2019-05-27') |
+-----------------------------+
| 2019-05-27 00:00:00         |
```

```
select timestamp('2019-05-27');
+-------------------------+
| timestamp('2019-05-27') |
+-------------------------+
|    2019-05-27 00:00:00 |
```

TIMESTAMPADD

```
TIMESTAMPADD(unit,interval,datetime_expr)
```

- Description: This function adds the interval specified by interval to the date or datetime expression specified by datetime_expr. unit specifies the unit of the interval.

  Valid values of unit: second, minute, hour, day, week, month, quarter, and year.
- Parameter types:

```
timestampadd(varchar, varchar, timestamp)
timestampadd(varchar, bigint, timestamp)
timestampadd(varchar, varchar, date)
timestampadd(varchar, bigint, date)
timestampadd(varchar, varchar, datetime)
timestampadd(varchar, bigint, datetime)
timestampadd(varchar, varchar, varchar)
timestampadd(varchar, bigint, varchar)
```

- Return value type: DATETIME.
- Example:

```
select timestampadd(second,'1',timestamp '2003-01-02 12:12:12')as result;
+---------------------+
```

```
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,1,timestamp '2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,'1',date '2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 00:00:01 |
```

```
select timestampadd(second,1,date '2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 00:00:01 |
```

```
select timestampadd(second,'1',datetime '2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,1,datetime '2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,'1','2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,1,'2003-01-02 12:12:12')as result;
+---------------------+
| result              |
+---------------------+
| 2003-01-02 12:12:13 |
```

```
select timestampadd(second,1,'2003-01-02 12:12:12');
+-----------------------------------------------+
| timestampadd('second', 1, '2003-01-02 12:12:12') |
+-----------------------------------------------+
| 2003-01-02 12:12:13                           |
```

```
select timestampadd(minute,8820,'2019-08-24 09:00:00');
+-------------------------------------------------+
| timestampadd('MINUTE', 8820, '2019-08-24 09:00:00') |
+-------------------------------------------------+
```

```
| 2019-08-30 12:00:00
```

```
select timestampadd(hour,1,'2003-01-02 12:12:12');
+-----------------------------------------------+
| timestampadd('hour', 1, '2003-01-02 12:12:12') |
+-----------------------------------------------+
| 2003-01-02 13:12:12                           |
```

```
select timestampadd(day,1,'2003-01-02 12:12:12');
+----------------------------------------------+
| timestampadd('day', 1, '2003-01-02 12:12:12') |
+----------------------------------------------+
| 2003-01-03 12:12:12                          |
```

```
select timestampadd(week,1,'2003-01-02 12:12:12');
+-----------------------------------------------+
| timestampadd('week', 1, '2003-01-02 12:12:12') |
+-----------------------------------------------+
| 2003-01-09 12:12:12                           |
```

```
select timestampadd(month,1,'2003-01-02 12:12:12');
+-----------------------------------------------+
| timestampadd('month', 1, '2003-01-02 12:12:12') |
+-----------------------------------------------+
| 2003-02-02 12:12:12                           |
```

```
select timestampadd(year,1,'2003-01-02 12:12:12');
+----------------------------------------------+
| timestampadd('year', 1, '2003-01-02 12:12:12') |
+----------------------------------------------+
| 2004-01-02 12:12:12                          |
```

```
select timestampadd(quarter,1,'2003-01-02 12:12:12');
+------------------------------------------------+
| timestampadd('quarter', 1, '2003-01-02 12:12:12') |
+------------------------------------------------+
| 2003-04-02 12:12:12                            |
```

TIMESTAMPDIFF

```
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
```

- Description: This function subtracts the interval specified by `datetime_expr2` from the date or datetime expression specified by `datetime_expr1`. `unit` specifies the unit of the result.

  Valid values of `unit`: `second`, `minute`, `hour`, `day`, `week`, `month`, `quarter`, and `year`.

  Use this function in the same way as the TIMESTAMPADD function.

- Parameter types:

```
timestampdiff(varchar, timestamp, timestamp)
timestampdiff(varchar, date, date)
timestampdiff(varchar, datetime, datetime)
```

```
timestampdiff(varchar, varchar, varchar)
```

- Return value type: BIGINT.
- Example:

```
select timestampdiff(second,datetime '2003-02-01 10:12:13',datetime '2003-05-01 10
:12:13')as result;
+---------+
| result  |
+---------+
| 7689600 |
```

```
select timestampdiff(minute,datetime '2003-02-01 10:12:13',datetime '2003-05-01 10
:12:13')as result;
+--------+
| result |
+--------+
| 128160 |
```

```
select timestampdiff(hour,datetime '2003-02-01 10:12:13',datetime '2003-05-01 10:
12:13')as result;
+--------+
| result |
+--------+
|   2136 |
```

```
select timestampdiff(day,timestamp '2003-02-01',timestamp '2003-05-01')as result;
+--------+
| result |
+--------+
|    89 |
```

```
select timestampdiff(day,date '2003-02-01',date '2003-05-01');
+-------------------------------------------------------------+
| timestampdiff('day', DATE '2003-02-01', DATE '2003-05-01') |
+-------------------------------------------------------------+
|                                          89 |
```

```
select timestampdiff(day,datetime '2003-02-01 10:12:13',datetime '2003-05-01 10:12
:13')as result;
+--------+
| result |
+--------+
|    89 |
```

```
select timestampdiff(day,'2003-02-01','2003-05-01');
+-----------------------------------------------+
| timestampdiff('day', '2003-02-01', '2003-05-01') |
+-----------------------------------------------+
|                                    89 |
```

```
select timestampdiff(week,'2003-02-01','2003-05-01');
+------------------------------------------------+
| timestampdiff('week', '2003-02-01', '2003-05-01') |
+------------------------------------------------+
```

```
|                                    12 |
```

```
select timestampdiff(quarter,'2003-02-01','2003-05-01');
+-------------------------------------------------------+
| timestampdiff('quarter', '2003-02-01', '2003-05-01') |
+-------------------------------------------------------+
|                                     1 |
```

```
select timestampdiff(month,'2003-02-01','2003-05-01');
+-------------------------------------------------------+
| timestampdiff('month', '2003-02-01', '2003-05-01') |
+-------------------------------------------------------+
|                                     3 |
```

```
select timestampdiff(year,datetime '2003-02-01 10:12:13',datetime '2001-05-01 10:12
:13')as result;
+--------+
| result |
+--------+
|    -1 |
```

TO_DAYS

```
TO_DAYS(date)
```

- Description: This function returns the number of days since year `0` based on the date specified by `date`.
- Parameter types:

```
to_days(date)
to_days(time)
to_days(varchar)
to_days(timestamp)
to_days(datetime)
```

- Return value type: BIGINT.
- Example:

```
select to_days(date '2018-12-12');
+---------------------------+
| to_days(DATE '2018-12-12') |
+---------------------------+
|                    737405 |
```

```
select to_days(time '12:12:12');
+-------------------------+
| to_days(TIME '12:12:12') |
+-------------------------+
|                  737572 |
```

```
select to_days(now());
+----------------+
| to_days(now()) |
+----------------+
```

```
|       737572 |
```

The preceding query is equivalent to to_days(curdate()).

```
select to_days(curdate());
+-------------------+
| to_days(curdate()) |
+-------------------+
|          737573 |
```

```
select to_days(datetime '2019-09-08 12:12:12');
+----------------------------------------+
| to_days(DATETIME '2019-09-08 12:12:12') |
+----------------------------------------+
|                          737675 |
```

```
select to_days('2019-09-08 12:12:12');
+-------------------------------+
| to_days('2019-09-08 12:12:12') |
+-------------------------------+
|                    737675 |
```

```
select to_days(timestamp '2019-09-08 12:12:12');
+----------------------------------------+
| to_days(TIMESTAMP '2019-09-08 12:12:12') |
+----------------------------------------+
|                          737675 |
```

TO_SECONDS

```
TO_SECONDS(expr)
```

- Description: This function returns the number of seconds that have elapsed since year 0 based on the time specified by expr.
- Parameter types:

```
to_seconds(date)
to_seconds(datetime)
to_seconds(timestamp)
to_seconds(varchar)
to_seconds(time)
```

- Return value type: BIGINT.
- Example:

```
select to_seconds(date '2019-09-08');
+-------------------------------+
| to_seconds(DATE '2019-09-08') |
+-------------------------------+
|                63735120000 |
```

```
select to_seconds(datetime '2019-09-08 09:09:00');
+----------------------------------------+
| to_seconds(DATETIME '2019-09-08 09:09:00') |
+----------------------------------------+
```

```
|                        63735152940 |
```

```
select to_seconds(timestamp '2019-09-08 09:09:00');
+-----------------------------------------+
| to_seconds(TIMESTAMP '2019-09-08 09:09:00') |
+-----------------------------------------+
|                        63735152940 |
```

If you execute the following SQL statement, the system adds curdate() to '09:09:00'.

```
select to_seconds(time '09:09:00');
+----------------------------+
| to_seconds(TIME '09:09:00') |
+----------------------------+
|             63726253740 |
```

```
select to_seconds('2019-09-08');
+-------------------------+
| to_seconds('2019-09-08') |
+-------------------------+
|           63735120000 |
```

UNIX_TIMESTAMP

```
UNIX_TIMESTAMP([date])
```

- Description: UNIX_TIMESTAMP() returns the timestamp for the current time. The timestamp follows the UNIX time format. It is the number of seconds that have elapsed since '1970-01-01 00:00:00' UTC. UNIX_TIMESTAMP(date) returns the timestamp for a date. It is the number of seconds that have elapsed since '1970-01-01 00:00:00' UTC.
- Parameter types:

```
unix_timestamp()
unix_timestamp(varchar)
unix_timestamp(timestamp)
unix_timestamp(date)
unix_timestamp(datetime)
```

- Return value type: BIGINT.
- Example:

```
select unix_timestamp();
+-----------------+
| unix_timestamp() |
+-----------------+
|      1558935850 |
```

```
select unix_timestamp(timestamp '2019-09-08 12:12:12');
+-----------------------------------------------+
| unix_timestamp(TIMESTAMP '2019-09-08 12:12:12') |
+-----------------------------------------------+
```

```
|                          1567915932 |
```

```
select unix_timestamp(date '2019-09-08');
+----------------------------------+
| unix_timestamp(DATE '2019-09-08') |
+----------------------------------+
|                 1567872000 |
```

```
select unix_timestamp(datetime '2019-09-08 12:12:12');
+-----------------------------------------------+
| unix_timestamp(DATETIME '2019-09-08 12:12:12') |
+-----------------------------------------------+
|                          1567915932 |
```

```
select unix_timestamp('2019-09-08 12:12:12');
+---------------------------------------+
| unix_timestamp('2019-09-08 12:12:12') |
+---------------------------------------+
|                    1567915932 |
```

UTC_DATE

```
UTC_DATE()
```

- Description: This function returns the UTC date.
- Return value type: VARCHAR.
- Example:

```
select utc_date();
+------------+
| utc_date() |
+------------+
| 2019-05-27 |
```

UTC_TIME

```
UTC_TIME()
```

- Description: This function returns the UTC time.
- Return value type: VARCHAR.
- Example:

```
select utc_time();
+------------+
| utc_time() |
+------------+
```

```
| 05:53:19  |
```

UTC_TIMESTAMP

```
utc_timestamp()
```

- Description: This function returns the UTC timestamp.
- Return value type: VARCHAR.
- Example:

```
select utc_timestamp();
+---------------------+
| utc_timestamp()     |
+---------------------+
| 2019-05-27 05:55:15 |
```

WEEK

```
WEEK(date[,mode])
```

- Description: This function returns the week number for the date specified by `date`, which is the week to which `date` belongs in the year.
  - `date` specifies the date for which you want to obtain the week number.
  - `mode` is an optional parameter that specifies the logic for calculating the week number. It allows you to specify whether the week starts from Monday or Sunday. The return value ranges from `0` to `52` or from `0` to `53`. The following table describes the formats that `mode` supports.

0	Sunday	0 to 53
1	Monday	0 to 53
2	Sunday	1 to 53
3	Monday	1 to 53
4	Sunday	0 to 53
5	Monday	0 to 53
6	Sunday	1 to 53
7	Monday	1 to 53

- Parameter types:

```
week(varchar)
week(varchar, bigint)
week(date)
week(date, bigint)
week(datetime)
week(datetime, bigint)
```

```
week(timestamp)
week(timestamp, bigint)
```

- Return value type: BIGINT.
- Example:

```
select week('2019-05-27');
+------------------+
| week('2019-05-27') |
+------------------+
|               21 |
```

```
select week('2008-02-20',1);
+----------------------+
| week('2008-02-20', 1) |
+----------------------+
|                    8 |
```

```
select week(date '2008-02-20');
+------------------------+
| week(DATE '2008-02-20') |
+------------------------+
|                     7 |
```

```
select week(date '2008-02-20',1);
+--------------------------+
| week(DATE '2008-02-20', 1) |
+--------------------------+
|                        8 |
```

```
select week(datetime '2008-02-20 00:00:00',1);
+-----------------------------------------+
| week(DATETIME '2008-02-20 00:00:00', 1) |
+-----------------------------------------+
|                                       8 |
```

```
select week(datetime '2008-02-20 00:00:00');
+--------------------------------------+
| week(DATETIME '2008-02-20 00:00:00') |
+--------------------------------------+
|                                    7 |
```

```
select week(timestamp '2008-02-20 00:00:00');
+---------------------------------------+
| week(TIMESTAMP '2008-02-20 00:00:00') |
+---------------------------------------+
|                                     7 |
```

```
select week(timestamp '2008-02-20 00:00:00',1);
+-----------------------------------------+
| week(TIMESTAMP '2008-02-20 00:00:00', 1) |
+-----------------------------------------+
```

```
|                                 8 |
```

WEEKDAY

```
WEEKDAY(date)
```

- Description: This function returns the weekday for the date specified by `date`. The result is an integer indicating the weekday. The mapping is as follows: `0 = Monday, 1 = Tuesday, ... 6 = Sunday`.
- Parameter types:

```
weekday(timestamp)
weekday(datetime)
weekday(date)
weekday(varchar)
```

- Return value type: BIGINT.
- Example:

```
select weekday(timestamp '2019-05-27 00:09:00');
+-----------------------------------------+
| weekday(TIMESTAMP '2019-05-27 00:09:00') |
+-----------------------------------------+
|                                       0 |
```

```
select weekday(datetime '2019-05-27 00:09:00');
+-----------------------------------------+
| weekday(DATETIME '2019-05-27 00:09:00') |
+-----------------------------------------+
|                                       0 |
```

```
select weekday(date '2019-05-27 00:09:00');
+-------------------------------------+
| weekday(DATE '2019-05-27 00:09:00') |
+-------------------------------------+
|                                   0 |
```

```
select weekday('2019-05-27');
+----------------------+
| weekday('2019-05-27') |
+----------------------+
|                    0 |
```

WEEKOFYEAR

```
WEEKOFYEAR(date)
```

- Description: This function returns the calendar week for the date specified by `date`. Valid values: `[1, 53]`.
- Parameter types:

```
weekofyear(timestamp)
```

```
weekofyear(datetime)
weekofyear(date)
weekofyear(varchar)
```

- Return value type: BIGINT.
- Example:

```
select weekofyear(timestamp '2019-05-27 09:00:00');
+-------------------------------------------+
| weekofyear(TIMESTAMP '2019-05-27 09:00:00') |
+-------------------------------------------+
|                                        22 |
```

```
select weekofyear(datetime '2019-05-27 09:00:00');
+-----------------------------------------+
| weekofyear(DATETIME '2019-05-27 09:00:00') |
+-----------------------------------------+
|                                      22 |
```

```
select weekofyear(date '2019-05-27');
+-------------------------------+
| weekofyear(DATE '2019-05-27') |
+-------------------------------+
|                            22 |
```

```
select weekofyear('2019-05-27');
+--------------------------+
| weekofyear('2019-05-27') |
+--------------------------+
|                       22 |
```

YEAR

```
YEAR(date)
```

- Description: This function returns the year part of the date specified by date.
- Parameter types:

```
year(timestamp)
year(datetime)
year(date)
year(time)
year(varchar)
```

- Return value type: BIGINT.
- Example:

```
select year(timestamp '2019-05-27 00:00:00');
+---------------------------------------+
| year(TIMESTAMP '2019-05-27 00:00:00') |
+---------------------------------------+
|                                  2019 |
```

```
select year(datetime '2019-05-27 00:00:00');
+---------------------------------------+
```

```
| year(DATETIME '2019-05-27 00:00:00') |
+--------------------------------------+
|                                 2019 |
```

```
select year(date '2019-05-27');
+------------------------+
| year(DATE '2019-05-27') |
+------------------------+
|                   2019 |
```

If you execute the following SQL statement, the system adds `curdate` to `'00:00:00'` and returns the result as a string.

```
select year(time '00:00:00');
+----------------------+
| year(TIME '00:00:00') |
+----------------------+
|                 2019 |
```

```
select year('2019-05-27');
+--------------------+
| year('2019-05-27') |
+--------------------+
|               2019 |
```

YEARWEEK

```
YEARWEEK(date)
YEARWEEK(date,mode)
```

- Description: This function the year and week of a date.

  Description: The year in the result may be different from the year in the date parameter for the first and the last week of the year.

  `mode` works in the same way as `mode` in the WEEK function. For the single-parameter syntax, the value of `mode` is `0`.

- Parameter types:

```
yearweek(timestamp)
yearweek(timestamp, bigint)
yearweek(datetime)
yearweek(datetime, bigint)
yearweek(date, bigint)
yearweek(date)
yearweek(varchar)
yearweek(varchar, bigint)
```

- Return value type: BIGINT.
- Example:

```
select yearweek(timestamp '2019-05-27 00:00:00');
+-----------------------------------------+
| yearweek(TIMESTAMP '2019-05-27 00:00:00') |
```

```
+-----------------------------------------+
|                                  201921 |
```

```
select yearweek(timestamp '2019-05-27 00:00:00',1);
+-------------------------------------------+
| yearweek(TIMESTAMP '2019-05-27 00:00:00', 1) |
+-------------------------------------------+
|                                    201922 |
```

```
select yearweek(datetime '2019-05-27 00:00:00');
+-----------------------------------------+
| yearweek(DATETIME '2019-05-27 00:00:00') |
+-----------------------------------------+
|                                  201921 |
```

```
select yearweek(datetime '2019-05-27 00:00:00',1);
+-------------------------------------------+
| yearweek(DATETIME '2019-05-27 00:00:00', 1) |
+-------------------------------------------+
|                                    201922 |
```

```
select yearweek(date '2019-05-27',1);
+--------------------------------+
| yearweek(DATE '2019-05-27', 1) |
+--------------------------------+
|                         201922 |
```

```
select yearweek(date '2019-05-27');
+-----------------------------+
| yearweek(DATE '2019-05-27') |
+-----------------------------+
|                      201921 |
```

```
select yearweek('2019-05-27');
+------------------------+
| yearweek('2019-05-27') |
+------------------------+
|                 201921 |
```

```
select yearweek('2019-05-27',1);
+--------------------------+
| yearweek('2019-05-27', 1) |
+--------------------------+
|                   201922 |
```

7 Numeric functions

AnalyticDB for MySQL supports the following numeric functions.

- ABS: returns the absolute value of an argument.
- ROUND: returns the result of rounding an argument.
- SQRT: returns the square root of an argument.
- LN: returns the natural logarithm of an argument.
- LOG: returns the logarithm of an argument.
- LOG2: returns the natural logarithm of an argument to the base 2.
- LOG10: returns the natural logarithm of an argument to the base 10.
- PI: returns the value of Pi.
- POWER/POW: returns the value of x raised to the power of y.
- RADIANS: converts degrees to radians.
- DEGREES: converts radians to degrees.
- SIGN: returns the value of the sign of an argument.
- CEILING/CEIL: returns the smallest integer value that is greater than the argument.
- FLOOR: returns the largest integer value that is less than the argument.
- EXP: returns the value of e (the base of the natural logarithm) raised to the power of x.
- COS: returns the cosine of an argument.
- ACOS: returns the arc cosine of an argument.
- TAN: returns the tangent of an argument.
- ATAN: returns the arc tangent of an argument.
- ATAN2: returns the arc tangent of the result of x divided by y.
- COT: returns the cotangent of an argument.
- ASIN: returns the arc sine of an argument.
- SIN: returns the sine of an argument.

ABS

```
abs(tinyint x)
abs(smallint x)
abs(int x)
abs(bigint x)
abs(float x)
abs(double x)
```

```
abs(decimal x)
```

- Description: This function returns the absolute value of x.
- Return value types: LONG, DECIMAL, and DOUBLE.
- Example:

```
select abs(4.5);
+----------+
| abs(4.5) |
+----------+
|      4.5 |
```

```
select abs(4);
+--------+
| abs(4) |
+--------+
|      4 |
```

ROUND

```
round(tinyint x)
round(smallint x)
round(int x)
round(bigint x)
round(float x)
round(double x)
round(x, d)
```

- Description: This function rounds the x argument. d specifies the number of decimal places. The default value of d is 0. The data type of the return value is the same as that of x.
  - If x is null, null is returned.
  - If d is greater than 0, the argument is rounded to the specified number of decimal places.
  - If d is equal to 0, the argument is rounded to the nearest integer.
  - If d is less than 0, the result is rounded to the number of digits left of the decimal point.
- Return value types: LONG, DECIMAL, and DOUBLE.
- Example:

```
select round(4);
+----------+
| round(4) |
+----------+
|        4 |
```

```
select round(4.5);
+------------+
```

```
| round(4.5) |
+------------+
|        5.0 |
```

```
select round(345.984, -1);
+-----------------------------+
| round(345.984, INTEGER '-1') |
+-----------------------------+
|                       350.0 |
```

SQRT

```
sqrt(double x)
```

- Description: This function returns the square root of x.
- Return value type: DOUBLE.
- Example:

```
select sqrt(4.222);
+-------------------+
| sqrt(4.222)       |
+-------------------+
| 2.054750593137766 |
```

LN

```
ln(double x)
```

- Description: This function returns the natural logarithm of x.
- Return value type: DOUBLE.
- Example:

```
select ln(2.718281828459045);
+----------------------+
| ln(2.718281828459045) |
+----------------------+
|                  1.0 |
```

LOG

```
log(double x)
log(double x, double y)
```

- Description: If this function is called with one argument, this function returns the natural logarithm of x. If called with two arguments, this function returns the logarithm of y to the base x.
- Return value type: DOUBLE.

- Example:

```
select log(16);
+------------------+
| log(16)          |
+------------------+
| 2.772588722239781 |
```

```
select log(10,100);
+--------------+
| log(10, 100) |
+--------------+
|          2.0 |
```

LOG2

```
log2(double x)
```

- Description: returns the natural logarithm of an argument to the base 2.
- Return value type: DOUBLE.
- Example:

```
select log2(8.7654);
+------------------+
| log2(8.7654)     |
+------------------+
| 3.131819928389146 |
```

LOG10

```
log10(double x)
```

- Description: This function returns the natural logarithm of an argument to the base 10.
- Return value type: DOUBLE.
- Example:

```
select log10(100.876);
+--------------------+
| log10(100.876)     |
+--------------------+
| 2.0037878529824615 |
```

PI

```
pi()
```

- Description: This function returns the value of Pi.
- Return value type: DOUBLE.
- Example:

```
select pi();
```

```
+------------------+
| pi()             |
+------------------+
| 3.141592653589793 |
```

POWER/POW

```
power(double x, double y)
pow(double x, double y)
```

- Description: This function returns the value of x raised to the power of y.
- Return value type: DOUBLE.
- Example:

```
select power(1.2,3.4);
+------------------+
| power(1.2, 3.4)  |
+------------------+
| 1.858729691979481 |
```

```
select pow(1.2,3.4);
+------------------+
| pow(1.2, 3.4)    |
+------------------+
| 1.858729691979481 |
```

RADIANS

```
radians(double x)
```

- Description: This function converts degrees to radians.
- Return value type: DOUBLE.
- Example:

```
select radians(60.0);
+-------------------+
| radians(60.0)     |
+-------------------+
| 1.0471975511965976 |
```

DEGREES

```
degrees(double x)
```

- Description: This function converts radians to degrees.
- Return value type: DOUBLE.
- Example:

```
select degrees(1.3);
+------------------+
| degrees(1.3)     |
+------------------+
```

```
| 74.48451336700703 |
```

SIGN

```
sign(smallint x)
sign(tinyint x)
sign(int x)
sign(bigint x)
sign(float x)
sign(double x)
sign(decimal x)
```

- Description: This function returns the sign of x.
- Return value type: LONG.
- Example:

```
select sign(12);
+----------+
| sign(12) |
+----------+
|       1 |
```

```
select sign(4.5);
+-----------+
| sign(4.5) |
+-----------+
|         1 |
```

CEILING/CEIL

```
ceiling(tinyint x)
ceiling(smallint x)
ceiling(int x)
ceiling(bigint x)
ceiling(float x)
ceiling(double x)
```

- Description: This function returns the smallest integer value that is greater than x.
- Return value types: LONG, DECIMAL, and DOUBLE.
- Example:

```
select ceiling(4);
+------------+
| ceiling(4) |
+------------+
|          4 |
```

```
select ceiling(2.3);
+--------------+
| ceiling(2.3) |
+--------------+
```

```
|       3.0 |
```

FLOOR

```
floor(tinyint x)
floor(smallint x)
floor(int x)
floor(bigint x)
floor(float x)
floor(double x)
```

- Description: This function returns the largest integer value that is less than x.
- Return value types: LONG, DECIMAL, and DOUBLE.
- Example:

```
select floor(4.5);
+------------+
| floor(4.5) |
+------------+
|       4.0 |
```

```
select floor(7);
+----------+
| floor(7) |
+----------+
|       7 |
```

EXP

```
exp(double x)
```

- Description: This function returns the value of e (the base of the natural logarithm) raised to the power of x.
- Return value type: DOUBLE.
- Example:

```
select exp(4.5);
+------------------+
| exp(4.5)         |
+------------------+
| 90.01713130052181 |
```

COS

```
cos(double x)
```

- Description: This function returns the cosine of x.
- Return value type: DOUBLE.
- Example:

```
select cos(1.3);
```

```
+--------------------+
| cos(1.3)           |
+--------------------+
| 0.26749882862458735 |
```

ACOS

```
acos(double x)
```

- Description: This function returns the arc cosine of x.

  If x is greater than 1 or x is less than -1, null is returned.
- Return value type: DOUBLE.
- Example:

```
select acos(0.5);
+--------------------+
| acos(0.5)          |
+--------------------+
| 1.0471975511965979 |
```

TAN

```
tan(double x)
```

- Description: This function returns the tangent of x.
- Return value type: DOUBLE.
- Example:

```
select tan(8);
+--------------------+
| tan(8)             |
+--------------------+
| -6.799711455220379 |
```

ATAN

```
atan(double x)
```

- Description: This function returns the arc tangent of x.
- Return value type: DOUBLE.
- Example:

```
select atan(0.5);
+--------------------+
| atan(0.5)          |
+--------------------+
```

```
| 0.4636476090008061 |
```

ATAN2

```
atan2(double x, double y)
atan(double x, double y)
```

- Description: This function returns the arc tangent of the result of x divided by y.
- Return value type: DOUBLE.
- Example:

```
select atan2(0.5,0.3);
+-------------------+
| atan2(0.5, 0.3)   |
+-------------------+
| 1.0303768265243125 |
```

```
select atan(0.5,0.3);
+-------------------+
| atan(0.5, 0.3)    |
+-------------------+
| 1.0303768265243125 |
```

COT

```
cot(double x)
```

- Description: This function returns the cotangent of x.
- Return value type: DOUBLE.
- Example:

```
select cot(1.234);
+--------------------+
| cot(1.234)         |
+--------------------+
| 0.35013639786701445 |
```

ASIN

```
asin(double x)
```

- Description: This function returns the arc sine of x.
- Return value type: DOUBLE.
- Example:

```
select asin(0.5);
+-------------------+
| asin(0.5)         |
+-------------------+
```

```
| 0.5235987755982989 |
```

SIN

```
sin(double x)
```

- Description: This function returns the sine of x.
- Return value type: DOUBLE.
- Example:

```
select sin(1.234);
+-------------------+
| sin(1.234)        |
+-------------------+
| 0.9438182093746337 |
```

8 String functions

AnalyticDB for MySQL supports the following string functions.

- ASCII: returns the ASCII value of the leftmost character of a character or a string.
- BIN: returns the binary string of an integer.
- BIT_LENGTH: returns the length of the str argument, measured in bits.
- CHAR: returns a string of ASCII values of a specified integer.
- CHAR_LENGTH/CHARACTER_LENGTH: returns the length of a string, measured in characters.
- CONCAT: concatenates strings.
- CONCAT_WS: concatenates strings and separates them with delimiters.
- ELT: returns the string specified by the integer N.
- EXPORT_SET: returns a combined string based on the value of bits in bits field.
- FIELD: returns the index position of a specified string in a string list.
- FIND_IN_SET: returns the position of a character or string in another string.
- FORMAT: formats the number N and returns a string.
- HEX: returns the hexadecimal string of an integer, or returns a string which consists of the hexadecimal values of all characters in the string.
- INSTR: returns the position of the first occurrence of a substring in a string.
- LEFT: returns the N leftmost characters of a string.
- LENGTH/OCTET_LENGTH: returns the length of a string.
- LIKE: simple pattern matching.
- LOCATE: returns the position of the first occurrence of a string in another string.
- LOWER/LCASE: converts a string to lowercase.
- LPAD: returns a string that is left-padded with another string.
- LTRIM: removes leading spaces of a string.
- MAKE_SET: returns a set of comma-separated strings.
- MID: returns a substring of a specified length, which starts from a specified position in a string. It has the same effect as SUBSTR/SUBSTRING.
- OCT: returns the octal string of an integer.
- POSITION: returns the position of the first occurrence of a substring in a string.
- REPEAT: repeats a string for count times.

- REPLACE: replaces part of the characters in another string with a specified string.
- REVERSE: reverses the characters in a string.
- RIGHT: returns the N rightmost characters of a string.
- RLIKE/REGEXP: matches the expression string with the pattern string. If the two strings match, 1 is returned. Otherwise, 0 is returned.
- RPAD: returns a string that is right-padded with another string.
- RTRIM: removes trailing spaces of a string.
- SPACE: returns a string that consists of a specified number of spaces.
- STRCMP: returns 0, 1, or -1 based on the values of the two strings.
- SUBSTR/SUBSTRING: returns a substring of a specified length starting from the specified position.
- SUBSTRING_INDEX: returns the substring before the last occurrence of the delim delimiter in the str string.
- TRIM: removes leading and trailing spaces of a string.
- UPPER/UCASE: converts a string to uppercase.

ASCII

```
ASCII(varchar str)
```

- Description: This function returns the decimal ASCII value of the leftmost character of the `str` argument or the `str` string.
- Return value type: BIGINT.
- Example:

```
select ascii('2');
+------------+
| ascii('2') |
+------------+
|         50 |
```

```
select ascii('dx');
+-------------+
| ascii('dx') |
+-------------+
```

```
|       100 |
```

BIN

```
BIN(bigint N)
```

- Description: This function returns the binary string of N.

  If N is null, null is returned.
- Return value type: VARCHAR.
- Example:

```
select bin(12);
+---------+
| bin(12) |
+---------+
| 1100    |
```

BIT_LENGTH

```
BIT_LENGTH(varchar str)
```

- Description: This function returns the length of the str argument, measured in bits.
- Return value type: BIGINT.
- Example:

```
select bit_length('text');
+--------------------+
| bit_length('text') |
+--------------------+
|                 32 |
```

```
select bit_length('China');
+--------------------+
| bit_length('China') |
+--------------------+
|                 48 |
```

CHAR

```
CHAR(bigint N1, bigint N2...)
```

- Description: This function returns a string of decimal ASCII values of the integers, such as N1 and N2.
- Return value type: VARBINARY.
- Example:

```
select char(97,110,97,108,121,116,105,99,100,98);
+-------------------------------------------------+
| char(97, 110, 97, 108, 121, 116, 105, 99, 100, 98) |
```

```
+-----------------------------------------------+
| analyticdb                                    |
```

CHAR_LENGTH/CHARACTER_LENGTH

```
CHAR_LENGTH(varchar str)
```

- Description: This function returns the length of the `str` string, measured in characters.
- Return value type: BIGINT.
- Example:

```
select char_length('China');
+--------------------+
| char_length('China') |
+--------------------+
|                  2 |
```

```
select char_length('abc');
+--------------------+
| char_length('abc') |
+--------------------+
|                  3 |
```

CONCAT

```
concat(varchar str1, ..., varchar strn)
```

- Description: This function concatenates strings. If any of the arguments is `null`, `null` is returned.
- Return value type: VARCHAR.
- Example:

```
select concat('aliyun', ', ', 'analyticdb');
+--------------------------------------+
| concat('aliyun', ', ', 'analyticdb') |
+--------------------------------------+
| aliyun, analyticdb                   |
```

```
select concat('abc',null,'def');
+---------------------------+
| concat('abc', null, 'def') |
+---------------------------+
```

```
| NULL                    |
```

CONCAT_WS

```
concat_ws(varchar separator, varchar str1, ..., varchar strn)
```

- Description: This function concatenates strings and separates them with delimiters. The first separator argument is the delimiter for the rest of the arguments. Arguments whose value is null are not included in the final string.
- Return value type: VARCHAR.
- Example:

```
select concat_ws(',', 'First name', 'Second name', 'Last Name')as result;
+---------------------------------+
| result                          |
+---------------------------------+
| First name,Second name,Last Name |
```

```
select concat_ws(',','First name',NULL,'Last Name')as result;
+----------------------+
| result               |
+----------------------+
| First name,Last Name |
```

ELT

```
ELT(bigint N, varchar str1, varchar str2, varchar str3,...)
```

- Description: This function returns the string specified by the integer N.

  If N is less than 1 or is greater than the number of the string arguments, null is returned.
- Return value type: VARCHAR.
- Example:

```
select elt(4, 'Aa', 'Bb', 'Cc', 'Dd');
+-------------------------------+
| elt(4, 'Aa', 'Bb', 'Cc', 'Dd') |
+-------------------------------+
| Dd                            |
```

EXPORT_SET

```
EXPORT_SET(bigint bits, varchar on, varchar off[, varchar separator[, bigint number_of_
bits]])
```

- Description: This function returns a string in which the on and off strings are placed based on the bits 0 or 1 from right to left (from low-order to high-order) in the binary value of the integer bits. The on string is placed for bit 1, and the off string is placed for bit 0. These strings are separated by delimiters. The default delimiter is a comma (,). The

number_of_bits argument specifies the number of bits that are checked. Default value: 64.

If the value of the number_of_bits argument is greater than 64, the argument is silently clipped to 64.

The same value is returned when the number_of_bits argument is set to -1 or 64.

- Return value type: VARCHAR.
- Example:

```
select export_set(5,'1','0',',',2);
+---------------------------------+
| export_set(5, '1', '0', ',', 2) |
+---------------------------------+
| 1,0                             |
```

```
select  export_set(5,'1','0',',',10);
+----------------------------------+
| export_set(5, '1', '0', ',', 10) |
+----------------------------------+
| 1,0,1,0,0,0,0,0,0,0              |
```

FIELD

```
field(varchar str, varchar str1, varchar str2, varchar str3,...)
```

- Description: This function returns the position of the str string in the strings, such as str1, str2, and str3. If the str string is not found, 0 is returned.
- Return value type: BIGINT.
- Example:

```
select field('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
+-------------------------------------------+
| field('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff') |
+-------------------------------------------+
|                                         2 |
```

FIND_IN_SET

```
FIND_IN_SET(varchar str, varchar strlist)
```

- Description: This function returns the position of str in the strlist string.

  If str is not found in strlist or strlist is empty, 0 is returned.

  If either str or strlist is null, null is returned.
- Return value type: BIGINT.

- Example:

```
select find_in_set('b','a,b,c,d');
+----------------------------+
| find_in_set('b', 'a,b,c,d') |
+----------------------------+
|                          2 |
```

FORMAT

```
format(double X, bigint D)
```

- Description: This function formats the integer X to the #,###,###.##format rounded to D decimal places and returns the result as a string.

  If D is 0, the result has no decimal point or fractional part.
- Return value type: BIGINT.
- Example:

```
select format(12332.123456, 4)as result1, format(12332.1,4)as result2, format(12332
.2,0)as result3;
+-------------+-------------+---------+
| result1     | result2     | result3 |
+-------------+-------------+---------+
| 12,332.1235 | 12,332.1000 | 12,332  |
```

HEX

```
HEX(bigint N)
HEX(varchar str)
```

- Description: This function returns the hexadecimal string of the integer N or returns a string consisting of the hexadecimal values of all characters in str.
- Return value type: VARCHAR.
- Example:

```
select hex(16);
+---------+
| hex(16) |
+---------+
| 10      |
```

```
select hex('16');
+-----------+
| hex('16') |
+-----------+
```

```
| 3136     |
```

INSTR

```
INSTR(varchar str, varchar substr)
```

- Description: This function returns the position of the first occurrence of the substr substring in the str string.
- Return value type: BIGINT.
- Example:

```
select instr('foobarbar', 'bar');
+--------------------------+
| instr('foobarbar', 'bar') |
+--------------------------+
|                        4 |
```

LEFT

```
LEFT(varchar str, bigint len)
```

- Description: This function returns the leftmost len characters of the str string.

  If str or len is null, null is returned.
- Return value type: VARCHAR.
- Example:

```
select left('foobarbar', 5);
+----------------------------+
| left('foobarbar', 5) |
+----------------------------+
| fooba                      |
```

LENGTH/OCTET_LENGTH

```
length(varchar str)
```

- Description: This function returns the length of the str string.
- Return value type: BIGINT.
- Example:

```
select length('aliyun');
+-----------------+
| length('aliyun') |
+-----------------+
```

```
|              6 |
```

LIKE

```
expression [ NOT ] LIKE pattern [ESCAPE 'escape_char']
```

- Description: The LIKE operator is used to match the expression string with the pattern string. If the two strings match, 1 is returned. Otherwise, 0 is returned.

  pattern can contain the following wildcard characters:

  - %: matches strings of any length.
  - _: matches a single character.

  escape_char is used to escape the wildcard characters and treat the percent signs (%) and underscores (_) in pattern following the escape character as a regular character instead of a wildcard.
- Return value type: BIGINT.
- Example:

```
    select 'David!' like 'David_' as result1, 'David!' not like 'David_' as result2,  'David!'
like '%D%v%' as result3;
    +---------+---------+---------+
    | result1 | result2 | result3 |
    +---------+---------+---------+
    |       1 |       0 |       1 |
```

```
    select 'David_' like 'David|_' ESCAPE '|';
    +-----------------------------------+
    | 'David_' LIKE 'David|_' ESCAPE '|' |
    +-----------------------------------+
    |                                 1 |
```

LOCATE

```
LOCATE(varchar substr, varchar str)
LOCATE(varchar substr, varchar str, bigint pos)
```

- Description: This function returns the position of the first occurrence of substr in the str string or returns the position of the first occurrence of the substr substring in the str string, starting from the specified position pos.

  If substr is not in str,0 is returned.

  If substr or str is null, null is returned.
- Return value type: BIGINT.
- Example:

```
    select locate('bar', 'foobarbar');
```

```
+---------------------------+
| locate('bar', 'foobarbar') |
+---------------------------+
|                         4 |
```

```
select locate('bar', 'foobarbar', 7);
+-----------------------------+
| locate('bar', 'foobarbar', 7) |
+-----------------------------+
|                           7 |
```

LOWER/LCASE

```
lower(varchar str)
```

- Description: This function converts letters in the str string to lowercase.
- Return value type: VARCHAR.
- Example:

```
select lower('Aliyun');
+-----------------+
| lower('Aliyun') |
+-----------------+
| aliyun          |
```

LPAD

```
LPAD(varchar str, bigint len, varchar padstr)
```

- Description: This function returns the str string, left-padded with the padstr string to a length of len characters.

  If str contains more than len characters, the return value will be shortened to len characters.
- Return value type: VARCHAR.
- Example:

```
select lpad('Aliyun',9,'#');
+-----------------------+
| lpad('Aliyun', 9, '#') |
+-----------------------+
| ###Aliyun
```

LTRIM

```
LTRIM(varchar str)
```

- Description: This function removes the leading spaces of the str string.
- Return value type: VARCHAR.

- Example:

```
select ltrim('  abc');
+---------------+
| ltrim('  abc') |
+---------------+
| abc           |
```

MAKE_SET

```
MAKE_SET(bits, str1, str2,...)
```

- Description: This function returns a set value (a string containing substrings separated by delimiters) consisting of strings that have the corresponding bit within the bit set.

  The `str1` string corresponds to bit `0`, the `str2` string corresponds to bit `1`, and so on for the other bits. Strings with a value of `null` are not included within the final string.
- Return value type: VARCHAR.
- Example:

```
select make_set(5,'hello','nice','world');
+---------------------------------------+
| make_set(5, 'hello', 'nice', 'world') |
+---------------------------------------+
| hello,world                           |
```

```
select make_set(1 | 4,'hello','nice',NULL,'world')as result;
+--------+
| result |
+--------+
| hello  |
```

MID

```
MID(varchar str, bigint pos, bigint len)
```

- Description: Same as SUBSTR/SUBSTRING, this function returns a substring that contains `len` characters in length from the `str` string, starting from the `pos` position.
- Return value type: VARCHAR.
- Example:

```
select mid('Quadratically',5,6);
+---------------------------+
| mid('Quadratically', 5, 6) |
+---------------------------+
| ratica                    |
```

```
select mid('Sakila', -5, 3);
+-------------------------------+
| mid('Sakila', INTEGER '-5', 3) |
+-------------------------------+
```

```
| aki                    |
```

OCT

```
OCT(bigint N)
```

- Description: This function returns the octal string of the integer N.

  If N is null, null is returned.
- Return value type: VARCHAR.
- Example:

```
select oct(12);
+---------+
| oct(12) |
+---------+
| 14
```

POSITION

```
POSITION(varchar substr IN varchar str)
```

- Description: This function returns the position of the first occurrence of the substr substring in the str string, starting from position 1. If substring is not found in str, 0 is returned.
- Return value type: BIGINT.
- Example:

```
select position('bar' in 'foobarbar');
+---------------------------+
| locate('bar', 'foobarbar') |
+---------------------------+
|                         4 |
```

REPEAT

```
REPEAT(varchar str, bigint count)
```

- Description: This function repeats the str string for count times.

  If count is less than 1, an empty string is returned.

  If str or count is null, null is returned.
- Return value type: VARCHAR.
- Example:

```
select repeat('a', 3);
+---------------+
| repeat('a', 3) |
```

```
+----------------+
| aaa            |
```

```
select  repeat('abc', null);
+---------------------+
| repeat('abc', null) |
+---------------------+
| NULL                |
```

```
select  repeat(null, 3);
+-----------------+
| repeat(null, 3) |
+-----------------+
| NULL            |
```

REPLACE

```
REPLACE(varchar str, varchar from_str, varchar to_str)
```

- Description: This function replaces all occurrences of the from_str string in the str string with the to_str string.
- Return value type: VARCHAR.
- Example:

```
select replace('WWW.aliyun.com', 'W', 'w');
+-------------------------------------+
| replace('WWW.aliyun.com', 'W', 'w') |
+-------------------------------------+
| www.aliyun.com                      |
```

REVERSE

```
REVERSE(varchar str)
```

- Description: This function returns the str string with the order of the characters reversed.
- Return value type: VARCHAR.
- Example:

```
select reverse('123456');
+-------------------+
| reverse('123456') |
+-------------------+
| 654321            |
```

RIGHT

```
RIGHT(varchar str, bigint len)
```

- Description: This function returns the rightmost len characters of the str string.

  If str or len is null, null is returned.

- Return value type: VARCHAR.
- Example:

```
select right('abc',3);
+-----------------------+
| presto_right('abc', 3) |
+-----------------------+
| abc                   |
```

RLIKE/REGEXP

```
expression  RLIKE pattern
expression  REGEXP pattern
```

- Description: This function matches the expression string with the pattern string. If the two strings match, 1 is returned. Otherwise, 0 is returned.

  If expression or pattern is null, null is returned.
- Return value type: BIGINT.
- Example:

```
select 'Michael!' regexp '.*';
+------------------------------+
| regexp_like('Michael!', '. *') |
+------------------------------+
|                            1 |
```

```
select 'new*\n*line' regexp 'new\\*.\\*line';
+-----------------------------------------+
| regexp_like('new*
*line', 'new\\*.\\*line') |
+-----------------------------------------+
|                                       0 |
```

```
select 'c' regexp '^[a-d]';
+---------------------------+
| regexp_like('c', '^[a-d]') |
+---------------------------+
|                         1 |
```

RPAD

```
RPAD(varchar str, bigint len, varchar padstr)
```

- Description: This function returns the str string, right-padded with the padstr string to a length of len characters.

  If str contains more than len characters, the return value is shortened to len characters.
- Return value type: VARCHAR.

- Example:

```
select rpad('Aliyun',9,'#');
+-----------------------+
| rpad('Aliyun', 9, '#') |
+-----------------------+
| Aliyun###             |
```

RTRIM

```
RTRIM(varchar str)
```

- Description: This function removes the trailing spaces of the `str` string.
- Return value type: VARCHAR.
- Example:

```
select rtrim('barbar   ');
+--------------------+
| rtrim('barbar   ') |
+--------------------+
| barbar             |
```

SPACE

```
SPACE(bigint N)
```

- Description: This function returns a string that consists of a specified number of spaces.
- Return value type: VARCHAR.
- Example:

```
select concat("#", space(6), "#");
+----------------------------+
| concat('#', space(6), '#') |
+----------------------------+
| #      #                   |
```

STRCMP

```
STRCMP(varchar str1, varchar str2)
```

- Description: If the `str1` string is the same as the `str1` string, `0` is returned. If the `str1` string is smaller than the `str2` string based on the current sort order, `-1` is returned. Otherwise, `1` is returned.
- Return value type: BIGINT.
- Example:

```
select strcmp('text', 'text2');
+-------------------------+
| strcmp('text', 'text2') |
+-------------------------+
```

```
|                -1 |
```

SUBSTR/SUBSTRING

```
SUBSTRING(varchar str, bigint pos)
SUBSTRING(varchar str FROM pos)
SUBSTRING(varchar str, bigint pos, bigint len)
SUBSTRING(varchar str FROM pos FOR len)
```

- Description:
  - SUBSTRING(varchar str, bigint pos) or SUBSTRING(varchar str FROM pos): returns the substring starting from the pos position to the end of the string. If pos is less than 0, the substring will begin pos characters from the end of the string.
  - SUBSTRING(varchar str, bigint pos, bigint len) or SUBSTRING(varchar str FROM pos FOR len): returns a substring that contains len characters in length from the string, starting from the pos position. If pos is less than 0, the substring will begin pos characters from the end of the string.
- Return value type: VARCHAR.
- Example:

```
select substr('helloworld', 6);
+------------------------+
| substr('helloworld', 6) |
+------------------------+
| world
```

```
select substr('helloworld' from 6);
+------------------------+
| substr('helloworld', 6) |
+------------------------+
| world                  |
```

```
select substr('helloworld', 6, 3);
+---------------------------+
| substr('helloworld', 6, 3) |
+---------------------------+
| wor                       |
```

```
select substr('helloworld' from 6 for 3);
+---------------------------+
| substr('helloworld', 6, 3) |
+---------------------------+
```

```
| wor                  |
```

SUBSTRING_INDEX

```
SUBSTRING_INDEX(varchar str, varchar delim, bigint count)
```

- Description: This function returns the substring before the last occurrence of the delim delimiter in the str string.

  If count is greater than 0, this function returns all characters to the left of the last delim delimiter.

  If count is less than 0, this function returns all characters to the right of the last delim delimiter.

  The SUBSTRING_INDEX function performs a case-sensitive match when searching for the delim delimiter.
- Return value type: VARCHAR.
- Example:

```
select substring_index('www.aliyun.com', '.', 2);
+-----------------------------------------+
| substring_index('www.aliyun.com', '.', 2) |
+-----------------------------------------+
| www.aliyun                              |
```

TRIM

```
TRIM([remstr FROM] str)
TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)
```

- Description: This function removes the leading and trailing spaces or other specified characters from a string.
- Return value type: VARCHAR.
- Example:

```
select trim('  bar   ');
+------------------+
| trim('  bar   ') |
+------------------+
| bar              |
```

```
select trim(both 'x' from 'xxxbarxxx');
+------------------------+
| trim('x', 'xxxbarxxx') |
+------------------------+
| bar                    |
```

```
select trim(leading 'x' from 'xxxbarxxx');
+-------------------------+
```

```
| ltrim('x', 'xxxbarxxx') |
+-------------------------+
| barxxx                  |
```

```
select trim(trailing 'x' from 'xxxbarxxx');
+-------------------------+
| rtrim('x', 'xxxbarxxx') |
+-------------------------+
| xxxbar                  |
```

UPPER/UCASE

```
upper(varchar str)
```

- Description: This function converts the str string to uppercase.
- Return value type: VARCHAR.
- Example:

```
select upper('Aliyun');
+-----------------+
| upper('Aliyun') |
+-----------------+
| ALIYUN          |
```

9 Variable-length binary functions

AnalyticDB for MySQL supports the following variable-length binary functions.

- LENGTH: returns the length of an argument, measured in bytes.
- CHAR_LENGTH: returns the length of a string, measured in characters.
- ORD: returns the code of the character if the leftmost character of a string is a multibyte character.
- HEX: converts a string or number to a hexadecimal string and returns the result.
- UNHEX: interprets each pair of characters in an argument to a hexadecimal value, and then converts the hexadecimal value to bytes represented by numbers. The return value is a binary string.
- MD5: calculates the MD5 hash value of an argument.
- SHA1: calculates the SHA-1 checksum of a string.
- CRC32
- LOWER: returns an argument in lowercase.
- UPPER: returns an argument in uppercase.
- UPPER: returns an argument in uppercase.
- LTRIM: removes the spaces to the left of a string.
- RTRIM: removes the spaces to the right of a string.
- TRIM: removes the leading and trailing spaces of a string.
- COMPRESS: compresses a string and returns the result as a binary string.
- UNCOMPRESS: decompresses a string compressed by the COMPRESS() function.
- UNCOMPRESSED_LENGTH: returns the length of a string before compression.
- ENCRYPT: encrypts a string.
- AES_ENCRYPT: uses the AES algorithm to encrypt data.
- AES_DECRYPT
- SUBSTR: returns a specified substring.
- LEFT: returns the leftmost y characters of a string.
- RIGHT
- REPEAT: returns a string that repeats for a specified number of times.
- REVERSE: reverses characters in a string.
- SHA2: calculates the SHA-2 checksum of a string.

- LPAD: returns a string that is left-padded with another string.
- RPAD: returns a string that is right-padded with another string.
- TO_BASE64
- FROM_BASE64: decodes a Base64-encoded string and returns the result.

LENGTH

```
length(varbinary x)
```

- Description: This function returns the length of the x argument, measured in bytes.
- Return value type: LONG.
- Example:

```
select length(CAST('abc' AS VARBINARY));
+----------------------------------+
| length(CAST('abc' AS varbinary)) |
+----------------------------------+
|                                3 |
```

CHAR_LENGTH

```
char_length(varbinary x)
```

- Description: This function returns the length of the x string, measured in characters.
- Return value type: LONG.
- Example:

```
select char_length(CAST('abc' AS VARBINARY));
+---------------------------------------+
| char_length(CAST('abc' AS varbinary)) |
+---------------------------------------+
|                                     3 |
```

ORD

```
ord(varbinary x)
```

- Description: This function returns the code of the character if the leftmost character of the x string is a multibyte character.
- Return value type: LONG.
- Example:

```
select ord(CAST('China' AS VARBINARY));
+---------------------------------+
| ord(CAST('China' AS varbinary))   |
+---------------------------------+
```

```
|                    228 |
```

HEX

```
hex(varbinary x)
```

- Description: This function converts the x string or number to a hexadecimal string.
- Return value type: VARCHAR.
- Example:

```
select hex(CAST('China' AS VARBINARY));
+---------------------------------+
| hex(CAST('China' AS varbinary))  |
+---------------------------------+
| E4B8ADE59BBD                    |
```

UNHEX

```
unhex(varbinary x)
```

- Description: This function interprets each pair of characters in the x argument as a hexadecimal value, and then converts the hexadecimal value to bytes represented by numbers. The return value is a binary string.
- Return value type: VARBINARY.
- Example:

```
select unhex(CAST('China' AS VARBINARY));
+-----------------------------------+
| unhex(CAST('China' AS varbinary))  |
+-----------------------------------+
| NULL                              |
```

MD5

```
md5(varbinary x)
```

- Description: This function returns the MD5 hash value of the x argument.
- Return value type: VARCHAR.
- Example:

```
select md5(CAST('China' AS VARBINARY));
+---------------------------------+
| md5(CAST('China' AS varbinary))  |
+---------------------------------+
```

```
| c13dceabcb143acd6c9298265d618a9f |
```

SHA1

```
sha1(varbinary x)
```

- Description: This function calculates the SHA-1 checksum of the x string.
- Return value type: VARCHAR.
- Example:

```
select sha1(CAST('China' AS VARBINARY));
+------------------------------------------+
| sha1(CAST('China' AS varbinary))         |
+------------------------------------------+
| 101806f57c322fb403a9788c4c24b79650d02e77 |
```

CRC32

```
crc32(varbinary x)
```

- Description: This function returns the Cyclic Redundancy Check (CRC) code of the x argument.
- Return value type: LONG.
- Example:

```
select crc32(CAST('China' AS VARBINARY));
+-----------------------------------+
| crc32(CAST('China' AS varbinary))   |
+-----------------------------------+
|                       737014929 |
```

LOWER

```
lower(varbinary x)
```

- Description: This function returns the x argument in lowercase.
- Return value type: VARBINARY.
- Example:

```
select lower(CAST('ABC' AS VARBINARY));
+--------------------------------+
| lower(CAST('ABC' AS varbinary)) |
+--------------------------------+
```

```
| abc                          |
```

UPPER

```
upper(varbinary x)
```

- Description: This function returns the x argument in uppercase.
- Return value type: VARBINARY.
- Example:

```
select upper(CAST('abc' AS VARBINARY));
+--------------------------------+
| upper(CAST('abc' AS varbinary)) |
+--------------------------------+
| ABC                          |
```

LTRIM

```
ltrim(varbinary x)
```

- Description: This function removes space characters to the left of the x string.
- Return value type: VARBINARY.
- Example:

```
select ltrim(CAST('  China  ' AS VARBINARY));
+---------------------------------------+
| ltrim(CAST('  China  ' AS varbinary))  |
+---------------------------------------+
| China                              |
```

RTRIM

```
rtrim(varbinary x)
```

- Description: This function removes space characters to the left of the x string.
- Return value type: VARBINARY.
- Example:

```
select rtrim(CAST('  China  ' AS VARBINARY));
+---------------------------------------+
| rtrim(CAST('  China  ' AS varbinary))  |
+---------------------------------------+
|   China                            |
```

TRIM

```
trim(varbinary x)
```

- Description: This function removes space characters to the left and right of the x string.

- Return value type: VARBINARY.
- Example:

```
select trim(CAST('  China  ' AS VARBINARY));
  +--------------------------------------+
  | trim(CAST('  China  ' AS varbinary))  |
  +--------------------------------------+
  | China                                |
```

COMPRESS

```
compress(varbinary x)
```

- Description: This function compresses the x string and returns the result as a binary string.
- Return value type: VARBINARY.
- Example:

```
  select hex(compress(CAST('China' AS VARBINARY)));
  +-------------------------------------------+
  | hex(compress(CAST('China' AS varbinary)))  |
  +-------------------------------------------+
  | 06000000789C7BB263EDD3D97B01104C0487      |
```

UNCOMPRESS

```
uncompress(varbinary x)
```

- Description: This function decompresses the x string compressed by the COMPRESS() function.
- Return value type: VARBINARY.
- Example:

```
  select uncompress(compress(CAST('China' AS VARBINARY)));
  +-------------------------------------------------+
  | uncompress(compress(CAST('China' AS varbinary)))  |
  +-------------------------------------------------+
  | China                                           |
```

UNCOMPRESSED_LENGTH

```
uncompressed_length(varbinary x)
```

- Description: This function returns the length of the x string before compression.
- Return value type: LONG.
- Example:

```
  select uncompressed_length(compress(CAST('China' AS VARBINARY)));
  +-----------------------------------------------------------+
```

```
| uncompressed_length(compress(CAST('China' AS varbinary)))  |
+------------------------------------------------------------+
|                                                          6 |
```

ENCRYPT

```
encrypt(varbinary x, varchar y)
```

- Description: This function encrypts the x argument with y as the salt value.
- Return value type: VARBINARY.
- Example:

```
select encrypt('abdABC123','key');
+---------------------------+
| encrypt('abdABC123', 'key') |
+---------------------------+
| kezazmcIo.aCw             |
```

AES_ENCRYPT

```
aes_encrypt(varbinary x, varchar y)
```

- Description: This function uses the AES algorithm to encrypt x with y as the key.
- Return value type: VARBINARY.
- Example:

```
select HEX(aes_encrypt(CAST('China' AS VARBINARY), '0123'));
+-----------------------------------------------------+
| HEX(aes_encrypt(CAST('China' AS VARBINARY), '0123'))  |
+-----------------------------------------------------+
| DFB166F0A03113AA848C0CE545D58757                    |
```

AES_DECRYPT

```
aes_decrypt(varbinary x, varchar y)
```

- Description: This function uses the AES algorithm to decrypt x with y as the key.
- Return value type: VARBINARY.
- Example:

```
select aes_decrypt(aes_encrypt(CAST('China' AS VARBINARY), '0123'),'0123');
+-------------------------------------------------------------------+
| aes_decrypt(aes_encrypt(CAST('China' AS varbinary), '0123'), '0123')  |
+-------------------------------------------------------------------+
| China                                                             |
```

SUBSTR

```
substr(varbinary x, bigint y, double z)
substr(varbinary x, double y, double z)
substr(varbinary x, bigint y, bigint z)
```

```
substr(varbinary x, double y, bigint z)
substr(varbinary x, double y)
substr(varbinary x, bigint y)
```

- Description: This function returns the specified substring.
- Return value type: VARBINARY.
- Example:

```
select HEX(substr(CAST('China' AS VARBINARY), 2));
+-------------------------------------------+
| HEX(substr(CAST('China' AS VARBINARY), 2))  |
+-------------------------------------------+
| B8ADE59BBD                                |
+-------------------------------------------+
```

```
+---------------------------------------------------+
| HEX(substr(CAST('China' AS VARBINARY), 2.7, 1.7))  |
+---------------------------------------------------+
| ADE5                                              |
+---------------------------------------------------+
```

LEFT

```
left(varbinary x, bigint y)
left(varbinary x, double y)
```

- Description: This function returns the leftmost y characters of the x string.
- Return value type: VARBINARY.
- Example:

```
select left(CAST('China' AS VARBINARY),1000);
+----------------------------------------+
| LEFT(CAST('China' AS VARBINARY), 1000)  |
+----------------------------------------+
| China                                  |
```

RIGHT

```
right(varbinary x, bigint y)
right(varbinary x, double y)
```

- Description: This function returns the rightmost y characters of the x string.
- Return value type: VARBINARY.
- Example:

```
select HEX(right(CAST('China' AS VARBINARY),1));
+-----------------------------------------+
| HEX(RIGHT(CAST('China' AS VARBINARY), 1))  |
+-----------------------------------------+
```

```
| BD                                  |
```

REPEAT

```
repeat(varbinary x, double y)
repeat(varbinary x, bigint y)
```

- Description: This function returns the x string that repeats for y times.
- Return value type: VARBINARY.
- Example:

```
select HEX(repeat(CAST('China' AS VARBINARY),1));
+-----------------------------------------+
| HEX(repeat(CAST('China' AS VARBINARY), 1))   |
+-----------------------------------------+
| E4B8ADE59BBD                            |
```

REVERSE

```
reverse(varbinary x)
```

- Description: This function reverses characters in the x string.
- Return value type: VARBINARY.
- Example:

```
select HEX(reverse(CAST('China' AS VARBINARY)));
+-----------------------------------------+
| HEX(reverse(CAST('China' AS VARBINARY)))   |
+-----------------------------------------+
| BD9BE5ADB8E4                            |
```

SHA2

```
sha2(varbinary x, integer y)
sha2(varbinary x, varchar y)
```

- Description: This function returns the SHA-2 checksum of the x string. You can choose to use SHA-224, SHA-256, SHA-384, and SHA-512. x is the plaintext string to be hashed. y is the length of bits required to represent the result, which must be 224, 256, 384, 512, or 0.
- Return value type: VARCHAR.
- Example:

```
select sha2(CAST('China' AS VARBINARY),256);
----------------+------------------------------------------------------------------
| sha2(CAST('China' AS varbinary), 256)                          |
----------------+------------------------------------------------------------------
```

```
| f0e9521611bb290d7b09b8cd14a63c3fe7cbf9a2f4e0090d8238d22403d35182 |
```

LPAD

```
lpad(varbinary x, bigint y, varchar z)
lpad(varbinary x, double y, varchar z)
```

- Description: This function returns the x string, left-padded with the z string to a length of y characters.

  If x is longer than y, the return value is shortened to y characters.
- Return value type: VARBINARY.
- Example:

```
select HEX(lpad(CAST('China' AS VARBINARY), 7,'-'));
+-----------------------------------------------+
| HEX(lpad(CAST('China' AS VARBINARY), 7, '-'))  |
+-----------------------------------------------+
| 2DE4B8ADE59BBD                                |
```

RPAD

```
rpad(varbinary x, bigint y, varchar z)
rpad(varbinary x, double y, varchar z)
```

- Description: This function returns the x string, right-padded with the z string to a length of y characters.

  If x is longer than y, the return value is shortened to y characters.
- Return value type: VARBINARY.
- Example:

```
select HEX(rpad(CAST('China' AS VARBINARY), 4.7,'x'));
+-----------------------------------------------+
| HEX(rpad(CAST('China' AS VARBINARY), 4.7, 'x'))  |
+-----------------------------------------------+
| E4B8ADE59B                                    |
```

TO_BASE64

```
to_base64(varbinary x)
```

- Description: This function returns the x string encoded in Base64 format.
- Return value type: VARCHAR.
- Example:

```
select to_base64(CAST('China' AS VARBINARY));
+---------------------------------------+
| to_base64(CAST('China' AS varbinary))  |
```

```
+----------------------------------------+
| 5Lit5Zu9                               |
```

FROM_BASE64

```
from_base64(varbinary x)
```

- Description: This function decodes the Base64-encoded x string and returns the result.
- Return value type: VARBINARY.
- Example:

```
select from_base64(to_base64(CAST('abc' AS VARBINARY)));
+-------------------------------------------------+
| from_base64(to_base64(CAST('abc' AS varbinary))) |
+-------------------------------------------------+
| abc                                             |
```

10 Window functions

AnalyticDB for MySQL supports the following window functions.

- Aggregate functions
- Sorting functions
  - CUME_DIST: returns the cumulative distribution of each value within a set of values.
  - RANK: returns the ranking of each value within a dataset.
  - DENSE_RANK: returns the ranking of each value within a set of values.
  - NTILE: divides data within each window partition into n buckets with bucket numbered from 1 to n.
  - ROW_NUMBER: returns a unique and sequential number for each row based on the sequence of the row within the window partition, starting from 1
  - PERCENT_RANK: returns the ranking percentage of each value in a dataset in the format of (r - 1)/(n - 1). r is the rank of the current row calculated by RANK(), and n is the total number of rows within the current window partition.
- Value functions
  - FIRST_VALUE: returns the value of the first row within the window partition.
  - LAST_VALUE: returns the value of the last row within the window partition.
  - LAG: returns the value of the row that precedes the current row by offset rows in the window.
  - LEAD: returns the value of the row that follows the current row by offset rows in the window.
  - NTH_VALUE: returns the value of the row by a specified number of offset rows in the window. The offset starts from 1.

Overview

Window functions calculate based on row data from the query result. Window functions run after the HAVING clause and before the ORDER BY clause. A window function can be triggered after you use an OVER clause to specify a window.

AnalyticDB for MySQL supports three types of window functions: aggregate functions, sorting functions, and value functions.

Syntax

```
function over (partition by a order by b RANGE|ROWS BETWEEN start AND end)
```

A window function contains the following parts:

- Partition rule: divides input rows to different partitions. The process is similar to the division process of the GROUP BY clause.
- Sorting rule: determines the order in which input rows are executed in the window function.
- Window frame: specifies the window boundary of the computed data.

  A window frame supports the RANGE and ROWS modes:

  - RANGE defines the range of column values.
  - ROWS defines the number of rows in a column.
  - For RANGE and ROWS, you can use BETWEEN start AND end to specify the boundary value. Valid values for the arguments in BETWEEN start AND end:

    ■ CURRENT ROW: the current row

    ■ N PRECEDING: the preceding n rows

    ■ UNBOUNDED PRECEDING: till the first row

    ■ N FOLLOWING: the following n rows

    ■ UNBOUNDED FOLLOWING: till the last row

For example, the following query calculates the partial sum of profit based on each row of data in the current window.

```
select year,country,profit,sum(profit) over (partition by country order by year ROWS
BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as slidewindow from testwindow;
+------+---------+--------+-------------+
| year | country | profit | slidewindow |
+------+---------+--------+-------------+
| 2001 | USA     |     50 |          50 |
| 2001 | USA     |   1500 |        1550 |
| 2000 | India   |     75 |          75 |
| 2000 | India   |     75 |         150 |
| 2001 | India   |     79 |         229 |
| 2000 | Finland |   1500 |        1500 |
| 2001 | Finland |     10 |        1510 |
```

The following query can only calculate the total sum of profit.

```
select country,sum(profit) over (partition by country) from testwindow;
+---------+---------------------------------------+
| country | sum(profit) OVER (PARTITION BY country) |
+---------+---------------------------------------+
| India   |                                   229 |
```

```
| India   |                    229 |
| India   |                    229 |
| USA     |                   1550 |
| USA     |                   1550 |
| Finland |                   1510 |
| Finland |                   1510 |
```

Precautions

Pay attention to the following requirements when you set boundary values:

- start cannot be UNBOUNDED FOLLOWING. Otherwise, the Window frame start cannot be UNBOUNDED FOLLOWING error will be prompted.
- end cannot be UNBOUNDED PRECEDING. Otherwise, the Window frame end cannot be UNBOUNDED PRECEDING error will be prompted.
- When start is CURRENT ROW and end is N PRECEDING, the Window frame starting from CURRENT ROW cannot end with PRECEDING error is prompted.
- When start is N FOLLOWING and end is N PRECEDING, the Window frame starting from FOLLOWING cannot end with PRECEDING error is prompted.
- When start is N FOLLOWING and end is CURRENT ROW, the Window frame starting from FOLLOWING cannot end with CURRENT ROW error is prompted.

When the window frame is in RANGE mode:

- When start or end is N PRECEDING, the Window frame RANGE PRECEDING is only supported with UNBOUNDED error is prompted.
- When start or end is N FOLLOWING, the Window frame RANGE FOLLOWING is only supported with UNBOUNDED error is prompted.

Preparations

The data from the testwindow table is used in the examples for the window functions in this topic.

```
create table testwindow(year int, country varchar(20), product varchar(20), profit int)
distributed by hash(year);
```

```
insert into testwindow values (2000,'Finland','Computer',1500);
insert into testwindow values (2001,'Finland','Phone',10);
insert into testwindow values (2000,'India','Calculator',75);
insert into testwindow values (2000,'India','Calculator',75);
insert into testwindow values (2001,'India','Calculator',79);
insert into testwindow values (2001,'USA','Calculator',50);
insert into testwindow values (2001,'USA','Computer',1500);
```

```
SELECT * FROM testwindow;
+------+---------+------------+--------+
| year | country | product    | profit |
```

```
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2001 | Finland | Phone      |     10 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Calculator |     75 |
| 2001 | India   | Calculator |     79 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   1500 |
```

Aggregate functions

All Aggregate functions can be used as window functions by adding the OVER clause. An aggregate function computes each row of data based on the rows within the current sliding window.

For example, the following query produces a rolling sum of order prices by date for each clerk:

```
SELECT clerk, orderdate, orderkey, totalprice,sum(totalprice) OVER (PARTITION BY clerk
ORDER BY orderdate) AS rolling_sum FROM orders ORDER BY clerk, orderdate, orderkey
```

CUME_DIST

```
CUME_DIST()
```

- Description: This function returns the cumulative distribution of each value in a set of values.

  Return result: The dataset after the window is ranked within the window partition, including the current row and the number of data rows preceding the current row. Any associated values in the sorting are calculated to the same distribution value.
- Return value type: DOUBLE.
- Example:

```
  select year,country,product,profit,cume_dist() over (partition by country order by
profit) as cume_dist from testwindow;
  +------+---------+------------+--------+--------------------+
  | year | country | product    | profit | cume_dist          |
  +------+---------+------------+--------+--------------------+
  | 2001 | USA     | Calculator |     50 |                0.5 |
  | 2001 | USA     | Computer   |   1500 |                1.0 |
  | 2001 | Finland | Phone      |     10 |                0.5 |
  | 2000 | Finland | Computer   |   1500 |                1.0 |
  | 2000 | India   | Calculator |     75 | 0.6666666666666666 |
  | 2000 | India   | Calculator |     75 | 0.6666666666666666 |
```

```
| 2001 | India   | Calculator |    79 |              1.0 |
```

RANK

```
RANK()
```

- Description: This function returns the rank of each value in a dataset.

  The rank value is the number of a row preceding the current row plus one but does not include the number of the current row. Therefore, tie values in the ordering may produce gaps in the sequence. The ranking is calculated for each window partition.
- Return value type: BIGINT.
- Example:

```
  select year,country,product,profit,rank() over (partition by country order by profit) as rank from testwindow;
  +------+---------+------------+--------+------+
  | year | country | product    | profit | rank |
  +------+---------+------------+--------+------+
  | 2001 | Finland | Phone      |    10 |    1 |
  | 2000 | Finland | Computer   |  1500 |    2 |
  | 2001 | USA     | Calculator |    50 |    1 |
  | 2001 | USA     | Computer   |  1500 |    2 |
  | 2000 | India   | Calculator |    75 |    1 |
  | 2000 | India   | Calculator |    75 |    1 |
  | 2001 | India   | Calculator |    79 |    3 |
```

DENSE_RANK

```
DENSE_RANK()
```

- Description: This function returns the ranking of each value in a set of values.

  DENSE_RANK() has similar features with RANK(), but the tie values of DENSE_RANK() do not produce gaps in the sequence.
- Return value type: BIGINT.
- Example:

```
  select year,country,product,profit,dense_rank() over (partition by country order by profit) as dense_rank from testwindow;
  +------+---------+------------+--------+------------+
  | year | country | product    | profit | dense_rank |
  +------+---------+------------+--------+------------+
  | 2001 | Finland | Phone      |    10 |          1 |
  | 2000 | Finland | Computer   |  1500 |          2 |
  | 2001 | USA     | Calculator |    50 |          1 |
  | 2001 | USA     | Computer   |  1500 |          2 |
  | 2000 | India   | Calculator |    75 |          1 |
  | 2000 | India   | Calculator |    75 |          1 |
```

```
| 2001 | India   | Calculator |    79 |        2 |
```

NTILE

```
NTILE(n)
```

- Description: This function divides data within each window partition into `n` buckets with bucket numbered from `1` to `n`.

  The maximum interval between bucket numbers is `1`. If the data rows within the window partition are not evenly distributed to each bucket, the remaining data will be distributed form the `first` bucket with `1` row of data for `each` bucket. For example, if there are six rows and four buckets, rows will be distributed to the buckets in the following manner: `1`, `1`, `2`, `2`, `3`, `and 4`.
- Return value type: BIGINT.
- Example:

```
  select year,country,product,profit,ntile(2) over (partition by country order by profit)
as ntile2 from testwindow;
  +------+---------+------------+--------+--------+
  | year | country | product    | profit | ntile2 |
  +------+---------+------------+--------+--------+
  | 2001 | USA     | Calculator |     50 |      1 |
  | 2001 | USA     | Computer   |   1500 |      2 |
  | 2001 | Finland | Phone      |     10 |      1 |
  | 2000 | Finland | Computer   |   1500 |      2 |
  | 2000 | India   | Calculator |     75 |      1 |
  | 2000 | India   | Calculator |     75 |      1 |
  | 2001 | India   | Calculator |     79 |      2 |
```

ROW_NUMBER

```
ROW_NUMBER()
```

- Description: This function returns a unique and sequential number for each row based on the sequence of the row within the window partition, starting from `1`.
- Return value type: BIGINT.
- Example:

```
  SELECT year, country, product, profit, ROW_NUMBER() OVER(PARTITION BY country)
AS row_num1 FROM testwindow;
  +------+---------+------------+--------+----------+
  | year | country | product    | profit | row_num1 |
  +------+---------+------------+--------+----------+
  | 2001 | USA     | Calculator |     50 |        1 |
  | 2001 | USA     | Computer   |   1500 |        2 |
  | 2000 | India   | Calculator |     75 |        1 |
  | 2000 | India   | Calculator |     75 |        2 |
  | 2001 | India   | Calculator |     79 |        3 |
  | 2000 | Finland | Computer   |   1500 |        1 |
```

```
| 2001 | Finland | Phone     |    10 |       2 |
```

PERCENT_RANK

```
PERCENT_RANK()
```

- Description: This function returns the ranking percentage of each data in a dataset, which is calculated by (r - 1)/(n - 1). r is the ranking of the current row by RANK() and n is the total number of rows within the current window partition.
- Return value type: DOUBLE.
- Example:

```
select year,country,product,profit,PERCENT_RANK() over (partition by country order by profit) as ntile3 from testwindow;
+------+---------+------------+--------+--------+
| year | country | product    | profit | ntile3 |
+------+---------+------------+--------+--------+
| 2001 | Finland | Phone      |    10 |    0.0 |
| 2000 | Finland | Computer   |  1500 |    1.0 |
| 2001 | USA     | Calculator |    50 |    0.0 |
| 2001 | USA     | Computer   |  1500 |    1.0 |
| 2000 | India   | Calculator |    75 |    0.0 |
| 2000 | India   | Calculator |    75 |    0.0 |
| 2001 | India   | Calculator |    79 |    1.0 |
```

FIRST_VALUE

```
FIRST_VALUE(x)
```

- Description: This function returns the value of the first row within the window partition.
- Return value type: the same as the input argument type.
- Example:

```
select year,country,product,profit,first_value(profit) over (partition by country order by profit) as firstValue from testwindow;
+------+---------+------------+--------+------------+
| year | country | product    | profit | firstValue |
+------+---------+------------+--------+------------+
| 2000 | India   | Calculator |    75 |         75 |
| 2000 | India   | Calculator |    75 |         75 |
| 2001 | India   | Calculator |    79 |         75 |
| 2001 | USA     | Calculator |    50 |         50 |
| 2001 | USA     | Computer   |  1500 |         50 |
| 2001 | Finland | Phone      |    10 |         10 |
| 2000 | Finland | Computer   |  1500 |         10 |
```

LAST_VALUE

```
LAST_VALUE(x)
```

- Description: This function returns the value of the last row within the window partition.
- Return value type: the same as the input argument type.

- Example:

```
select year,country,product,profit,last_value(profit) over (partition by country order by profit) as firstValue from testwindow;
+------+---------+------------+--------+------------+
| year | country | product    | profit | firstValue |
+------+---------+------------+--------+------------+
| 2001 | Finland | Phone      |     10 |         10 |
| 2000 | Finland | Computer   |   1500 |       1500 |
| 2001 | USA     | Calculator |     50 |         50 |
| 2001 | USA     | Computer   |   1500 |       1500 |
| 2000 | India   | Calculator |     75 |         75 |
| 2000 | India   | Calculator |     75 |         75 |
| 2001 | India   | Calculator |     79 |         79 |
```

LAG

```
LAG(x[, offset[, default_value]])
```

- Descriptions: This function returns the value of the row that precedes the current row by `offset` rows in the window.

  The starting offset value is `0`. The offset starts from the current data row. The offset can be a scalar expression. The default `offset` is `1`.

  If the value of offset is `null` or is greater than the window length, `default_value` is returned. If `default_value` is not specified, `null` is returned.
- Return value type: the same as the input argument type.
- Example:

```
select year,country,product,profit,lag(profit) over (partition by country order by profit) as lag from testwindow;
+------+---------+------------+--------+------+
| year | country | product    | profit | lag  |
+------+---------+------------+--------+------+
| 2001 | USA     | Calculator |     50 | NULL |
| 2001 | USA     | Computer   |   1500 |   50 |
| 2000 | India   | Calculator |     75 | NULL |
| 2000 | India   | Calculator |     75 |   75 |
| 2001 | India   | Calculator |     79 |   75 |
| 2001 | Finland | Phone      |     10 | NULL |
```

```
| 2000 | Finland | Computer   |  1500 |  10 |
```

LEAD

```
LEAD(x[,offset[, default_value]])
```

- Descriptions: This function returns the value of the row that follows the current row by offset rows in the window.

  The starting offset value is 0. The offset starts from the current data row. The offset can be a scalar expression. The default offset is 1.

  If the value of offset is null or is greater than the window length, default_value is returned. If default_value is not specified, null is returned.
- Return value type: the same as the input argument type.
- Example:

```
select year,country,product,profit,lead(profit) over (partition by country order by
profit) as lead from testwindow;
+------+---------+------------+--------+------+
| year | country | product    | profit | lead |
+------+---------+------------+--------+------+
| 2000 | India   | Calculator |     75 |   75 |
| 2000 | India   | Calculator |     75 |   79 |
| 2001 | India   | Calculator |     79 | NULL |
| 2001 | Finland | Phone      |     10 | 1500 |
| 2000 | Finland | Computer   |   1500 | NULL |
| 2001 | USA     | Calculator |     50 | 1500 |
| 2001 | USA     | Computer   |   1500 | NULL |
```

NTH_VALUE

```
NTH_VALUE(x, offset)
```

- Description: This function returns the value of the row by a specified number of offset rows in the window. The offset starts from 1.

  If offset is null or greater than the number of values in the window, null is returned. If offset is 0 or negative, an error is prompted.
- Return value type: the same as the input argument type.
- Example:

```
select year,country,product,profit,nth_value(profit,1) over (partition by country
order by profit) as nth_value from testwindow;
+------+---------+------------+--------+-----------+
| year | country | product    | profit | nth_value |
+------+---------+------------+--------+-----------+
| 2001 | Finland | Phone      |     10 |        10 |
| 2000 | Finland | Computer   |   1500 |        10 |
| 2001 | USA     | Calculator |     50 |        50 |
| 2001 | USA     | Computer   |   1500 |        50 |
```

```
| 2000 | India   | Calculator |    75 |       75 |
| 2000 | India   | Calculator |    75 |       75 |
| 2001 | India   | Calculator |    79 |       75 |
```

11 JSON functions

This topic describes the JSON functions used in AnalyticDB for MySQL.

JSON_EXTRACT

```
json_extract(json, jsonpath)
```

- Description: This function returns the value specified by `jsonpath` from `json`.
- Return value type: JSON.
- Example:

```
select json_extract('[10, 20, [30, 40]]', '$[1]');
 +------------------------------------------------+
 |                 20                 |
```

JSON _ARRAY_CONTAINS

```
json_array_contains(json, value)
```

- Description: This function determines whether `json` contains the value specified by `value`.
- Return value type: BOOLEAN.
- Example:

```
select json_array_contains('[1, 2, 3]', 2);
 +----------------------------------------+
 |              1              |
```

JSON_SIZE

```
json_size(json, jsonpath)
```

- Description: This function returns the size of `json`.
- Return value type: BIGINT.
- Example:

```
select json_size('{"x":{"a":1, "b": 2}}', '$.x') as result;
 +-------------------------------------------------------+
 |                    2                    |
```

```
select json_size('{"x": {"a": 1, "b": 2}}', '$.x.a') as result;
 +-----------------------------------------------------+
```

```
| 0 |
```

JSON_ARRAY_LENGTH

```
json_array_length(json)
```

- Description: This function returns the length of the `json` array.
- Return value type: BIGINT.
- Example:

```
select json_array_length('[1, 2, 3]')
 +----------------------------------+
 | 3 |
```

JSON_KEYS

```
json_keys(json, jsonpath)
```

- Description: This function obtains all the key values of `json` in a specified path.
- Return value type: JSON ARRAY.
- Example:

```
select json_keys(cast('{"a": 1, "b": {"c": 30}}' as json),'$.b')
 +-----------------------------------------------------------+
 | ["c"] |
```