Functions

ABS

This function works on a numeric column, converts it to a positive value, and returns the value as a real number.

```
ABS(column_name), ABS(column_name),...
```

```
Mach> CREATE TABLE abs_table (c1 integer, c2 double, c3 varchar(10));
Created successfully.

Mach> INSERT INTO abs_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO abs_table VALUES(2, 2.0, 'sqltest');
1 row(s) inserted.

Mach> INSERT INTO abs_table VALUES(3, 3.0, 'sqltest');
1 row(s) inserted.

Mach> SELECT ABS(c1), ABS(c2) from abs_table;
SELECT ABS(c1), ABS(c2) from abs_table;
ABS(c1)                     ABS(c2)
------------------------------------------------------------
3                           3
2                           2
1                           1
[3] row(s) selected.
```

ADD_TIME

This function performs a date and time operation on a given datetime column. Supports increment /decrement operations up to year, month, day, hour, minute, and second, and does not support operations on milli, micro, and nanoseconds. The Diff format is: "Year/Month/Day Hour:Minute:Second". Each item has a positive or negative value.

```
ADD_TIME(column,time_diff_format)
```

```
Mach> CREATE TABLE add_time_table (id INTEGER, dt DATETIME);
Created successfully.

Mach> INSERT INTO  add_time_table VALUES(1, TO_DATE('1999-11-11 1:2:3 4:5:
6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(2, TO_DATE('2000-11-11 1:2:3 4:5:
6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(3, TO_DATE('2012-11-11 1:2:3 4:5:
6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(4, TO_DATE('2013-11-11 1:2:3 4:5:
6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(5, TO_DATE('2014-12-30 11:22:33
444:555:666'));
1 row(s) inserted.
```

```
Mach> INSERT INTO  add_time_table VALUES(6, TO_DATE('2014-12-30 23:22:33
444:555:666'));
1 row(s) inserted.

Mach> SELECT ADD_TIME(dt, '1/0/0 0:0:0') FROM add_time_table;
ADD_TIME(dt, '1/0/0 0:0:0')
---------------------------------
2015-12-30 23:22:33 444:555:666
2015-12-30 11:22:33 444:555:666
2014-11-11 01:02:03 004:005:006
2013-11-11 01:02:03 004:005:006
2001-11-11 01:02:03 004:005:006
2000-11-11 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '0/0/0 1:1:1') FROM add_time_table;
ADD_TIME(dt, '0/0/0 1:1:1')
---------------------------------
2014-12-31 00:23:34 444:555:666
2014-12-30 12:23:34 444:555:666
2013-11-11 02:03:04 004:005:006
2012-11-11 02:03:04 004:005:006
2000-11-11 02:03:04 004:005:006
1999-11-11 02:03:04 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '1/1/1 0:0:0') FROM add_time_table;
ADD_TIME(dt, '1/1/1 0:0:0')
---------------------------------
2016-01-31 23:22:33 444:555:666
2016-01-31 11:22:33 444:555:666
2014-12-12 01:02:03 004:005:006
2013-12-12 01:02:03 004:005:006
2001-12-12 01:02:03 004:005:006
2000-12-12 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '-1/0/0 0:0:0') FROM add_time_table;
ADD_TIME(dt, '-1/0/0 0:0:0')
---------------------------------
2013-12-30 23:22:33 444:555:666
2013-12-30 11:22:33 444:555:666
2012-11-11 01:02:03 004:005:006
2011-11-11 01:02:03 004:005:006
1999-11-11 01:02:03 004:005:006
1998-11-11 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '0/0/0 -1:-1:-1') FROM add_time_table;
ADD_TIME(dt, '0/0/0 -1:-1:-1')
---------------------------------
2014-12-30 22:21:32 444:555:666
2014-12-30 10:21:32 444:555:666
2013-11-11 00:01:02 004:005:006
2012-11-11 00:01:02 004:005:006
2000-11-11 00:01:02 004:005:006
1999-11-11 00:01:02 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '-1/-1/-1 0:0:0') FROM add_time_table;
ADD_TIME(dt, '-1/-1/-1 0:0:0')
---------------------------------
2013-11-29 23:22:33 444:555:666
2013-11-29 11:22:33 444:555:666
2012-10-10 01:02:03 004:005:006
2011-10-10 01:02:03 004:005:006
1999-10-10 01:02:03 004:005:006
1998-10-10 01:02:03 004:005:006
[6] row(s) selected.
```

```
Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30
11:22:33 444:555:666'), '-1/-1/-1 0:0:0');
ID          DT
---------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
[2] row(s) selected.

Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30
11:22:33 444:555:666'), '-1/-2/-1 0:0:0');
ID          DT
---------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
4           2013-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT ADD_TIME(TO_DATE('2000-12-01 00:00:00 000:000:001'), '-1/0/0
0:0:-1') FROM add_time_table;
ADD_TIME(TO_DATE('2000-12-01 00:00:00 000:000:001'), '-1/0/0 0:0:-1')
-----------------------------------------
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
[6] row(s) selected.

Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30
11:22:33 444:555:666'), '-1/-2/-1 0:0:0');
ID          DT
---------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
4           2013-11-11 01:02:03 004:005:006
[3] row(s) selected.
```

AVG

This function is an aggregate function that operates on a numeric column and prints the average value of that column.

```
AVG(column_name)
```

```
Mach> CREATE TABLE avg_table (id1 INTEGER, id2 INTEGER);
Created successfully.

Mach> INSERT INTO avg_table VALUES(1, 1);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 3);
1 row(s) inserted.
```

```
Mach> INSERT INTO avg_table VALUES(null, 4);
1 row(s) inserted.

Mach> SELECT id1, AVG(id2) FROM avg_table GROUP BY id1;
id1         AVG(id2)
----------------------------------------
2               2
NULL            4
1               2
```

BITAND / BITOR

This function converts two input values to a 64-bit signed integer and returns the result of bitwise and/or. The input value must be an integer and the output value is a 64-bit signed integer.

For integer values less than 0, it is recommended to use only uinteger and ushort types, because different results may be obtained depending on the platform.

```
BITAND (<expression1>, <expression2>)
BITOR (<expression1>, <expression2>)
```

```
Mach> CREATE TABLE bit_table (i1 INTEGER, i2 UINTEGER, i3 FLOAT, i4 DOUBLE, i5 SHORT, i6 VARCHAR(10));
Created successfully.

Mach> INSERT INTO bit_table VALUES (-1, 1, 1, 1, 2, 'aaa');
1 row(s) inserted.

Mach> INSERT INTO bit_table VALUES (-2, 2, 2, 2, 3, 'bbb');
1 row(s) inserted.

Mach> SELECT BITAND(i1, i2) FROM bit_table;
BITAND(i1, i2)
----------------------
2
1
[2] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITAND(i2, 1) = 1;
I1          I2          I3                          I4                          I5          I6
------------------------------------------------------------------------------------------------------------------------
--
-1          1           1                           1                           2           aaa
[1] row(s) selected.

Mach> SELECT BITOR(i5, 1) FROM bit_table WHERE BITOR(i5, 1) = 3;
BITOR(i5, 1)
----------------------
3
3
[2] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITOR(i2, 1) = 1;
I1          I2          I3                          I4                          I5          I6
------------------------------------------------------------------------------------------------------------------------
--
-1          1           1                           1                           2           aaa
[1] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITAND(i3, 1) = 1;
I1          I2          I3                          I4                          I5          I6
------------------------------------------------------------------------------------------------------------------------
--
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.
```

```
Mach> SELECT * FROM bit_table WHERE BITAND(i4, 1) = 1;
I1          I2          I3                          I4                          I5          I6
------------------------------------------------------------------------------------------------------------------------
--
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT BITAND(i5, 1) FROM bit_table WHERE BITAND(i5, 1) = 1;
BITAND(i5, 1)
----------------------
1
[1] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITOR(i6, 1) = 1;
I1          I2          I3                          I4                          I5          I6
------------------------------------------------------------------------------------------------------------------------
--
[ERR-02037 : Function [BITOR] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT BITOR(i1, i2) FROM bit_table;
BITOR(i1, i2)
----------------------
-2
-1
[2] row(s) selected.

Mach> SELECT BITAND(i1, i3) FROM bit_table;
BITAND(i1, i3)
----------------------
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT BITOR(i1, i6) FROM bit_table;
BITOR(i1, i6)
----------------------
[ERR-02037 : Function [BITOR] argument data type is mismatched.]
[0] row(s) selected.
```

COUNT

This function is an aggregate function that obtains the number of records in a given column.

```
COUNT(column_name)
```

```
Mach> CREATE TABLE count_table (id1 INTEGER, id2 INTEGER);
Created successfully.

Mach> INSERT INTO count_table VALUES(1, 1);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 3);
```

```
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(null, 4);
1 row(s) inserted.

Mach> SELECT COUNT(*) FROM count_table;
COUNT(*)
----------------------
7
[1] row(s) selected.

Mach> SELECT COUNT(id1) FROM count_table;
COUNT(id1)
----------------------
6
[1] row(s) selected.
```

DATE_TRUNC

This function returns a given datetime value as a new datetime value that is displayed only up to 'time unit' and 'time range'.

```
DATE_TRUNC (field, date_val [, count])
```

```
Mach> CREATE TABLE trunc_table (i1 INTEGER, i2 DATETIME);
Created successfully.

Mach> INSERT INTO trunc_table VALUES (1, TO_DATE('1999-11-11 1:2:0 4:5:1'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (2, TO_DATE('1999-11-11 1:2:0 5:5:2'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (3, TO_DATE('1999-11-11 1:2:1 6:5:3'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (4, TO_DATE('1999-11-11 1:2:1 7:5:4'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (5, TO_DATE('1999-11-11 1:2:2 8:5:5'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (6, TO_DATE('1999-11-11 1:2:2 9:5:6'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (7, TO_DATE('1999-11-11 1:2:3 10:5:7'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (8, TO_DATE('1999-11-11 1:2:3 11:5:8'));
1 row(s) inserted.

Mach> SELECT COUNT(*), DATE_TRUNC('second', i2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
-----------------------------------------------------
2                    1999-11-11 01:02:00 000:000:000
2                    1999-11-11 01:02:01 000:000:000
2                    1999-11-11 01:02:02 000:000:000
2                    1999-11-11 01:02:03 000:000:000
[4] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('second', i2, 2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
-----------------------------------------------------
4                    1999-11-11 01:02:00 000:000:000
4                    1999-11-11 01:02:02 000:000:000
```

```
[2] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('nanosecond', i2, 2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
------------------------------------------------------
1                    1999-11-11 01:02:00 004:005:000
1                    1999-11-11 01:02:00 005:005:002
1                    1999-11-11 01:02:01 006:005:002
1                    1999-11-11 01:02:01 007:005:004
1                    1999-11-11 01:02:02 008:005:004
1                    1999-11-11 01:02:02 009:005:006
1                    1999-11-11 01:02:03 010:005:006
1                    1999-11-11 01:02:03 011:005:008
[8] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('nsec', i2, 1000000000) tm FROM trunc_table group by tm ORDER BY 2; --
same with DATE_TRUNC('sec', i2, 1)
COUNT(*)             tm
------------------------------------------------------
2                    1999-11-11 01:02:00 000:000:000
2                    1999-11-11 01:02:01 000:000:000
2                    1999-11-11 01:02:02 000:000:000
2                    1999-11-11 01:02:03 000:000:000
[4] row(s) selected.
```

The allowable time ranges for time units and time units are as follows.

> nanosecond, microsecond, milisecond supports in and after 5.5.6 version.

Time Unit	Time Range
nanosecond (nsec)	1000000000 (1 sec)
microsecond (usec)	60000000 (60 sec)
milisecond (msec)	60000 (60 sec)
second	86400
minute	1440
hour	24
day	1
month	1
year	1

For example, if you type in `DATE_TRUNC('second', time, 120)`, the value returned will be displayed **every two minutes** and is the same as `DATE_TRUNC('minute', time, 2)`.

DAYOFWEEK

This function returns a natural number representing the day of the week for a given datetime value.

Returns a semantically equivalent value for TO_CHAR (time, 'DAY'), but returns an integer here.

```
DAYOFWEEK(date_val)
```

The returned natural number represents the next day of the week.

Return Value	Day of Week
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

DECODE

This function compares the given Column value with Search, and returns the next return value if it is the same. If there is no satisfactory Search value, it returns the default value. If Default is omitted, NULL is returned.

```
DECODE(column, [search, return],.. default)
```

```
Mach> CREATE TABLE decode_table (id1 VARCHAR(11));
Created successfully.

Mach> INSERT INTO decode_table VALUES('decodetest1');
1 row(s) inserted.

Mach> INSERT INTO decode_table VALUES('decodetest2');
1 row(s) inserted.

Mach> SELECT id1, DECODE(id1, 'decodetest1', 'result1', 'decodetest2', 'result2', 'DEFAULT') FROM
decode_table;
id1          DECODE(id1, 'decodetest1', 'result1', 'decodetest2', 'result2', 'DEFAULT')
-------------------------------------------------------
decodetest2  result2
decodetest1  result1
[2] row(s) selected.

Mach> SELECT id1, DECODE(id1, 'codetest', 2, 99) FROM decode_table;
id1          DECODE(id1, 'codetest', 2, 99)
-----------------------------------------
decodetest2  99
decodetest1  99
[2] row(s) selected.

Mach> SELECT DECODE(id1, 'decodetest1', 2) FROM decode_table;
DECODE(id1, 'decodetest1', 2)
-------------------------------
NULL
2
[2] row(s) selected.

Mach> SELECT DECODE(id1, 'codetest', 2) FROM decode_table;
DECODE(id1, 'codetest', 2)
----------------------------
NULL
NULL
[2] row(s) selected.
```

FIRST / LAST

This function is an aggregate function that returns the specific value of the highest (or last) record in the sequence in which the 'reference value' in each group is in order.

- FIRST: Returns a specific value from the most advanced record in the sequence
- LAST: Returns a specific value from the last record in the sequence

```
FIRST(sort_expr, return_expr)
LAST(sort_expr, return_expr)
```

```
Mach> create table firstlast_table (id integer, name varchar(20), group_no integer);
Created successfully.
Mach> insert into firstlast_table values (1, 'John', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (2, 'Grey', 1);
1 row(s) inserted.
Mach> insert into firstlast_table values (5, 'Ryan', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (4, 'Andrew', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (7, 'Kyle', 1);
1 row(s) inserted.
Mach> insert into firstlast_table values (6, 'Ross', 1);
1 row(s) inserted.

Mach> select group_no, first(id, name) from firstlast_table group by group_no;
group_no    first(id, name)
------------------------------------
1           Grey
0           John
[2] row(s) selected.


Mach> select group_no, last(id, name) from firstlast_table group by group_no;
group_no    last(id, name)
------------------------------------
1           Kyle
0           Ryan
```

FROM_UNIXTIME

This function converts a 32-bit UNIXTIME value entered as an integer to a datetime datatype value. (UNIX_TIMESTAMP converts datetime data to 32-bit UNIXTIME integer data.)

```
FROM_UNIXTIME(unix_timestamp_value)
```

```
Mach> SELECT FROM_UNIXTIME(315540671) FROM TEST;
FROM_UNIXTIME(315540671)
---------------------------------
1980-01-01 11:11:11 000:000:000

Mach> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2001-01-01')) FROM unix_table;
FROM_UNIXTIME(UNIX_TIMESTAMP('2001-01-01'))
-----------------------------------------
2001-01-01 00:00:00 000:000:000
```

FROM_TIMESTAMP

This function takes a nanosecond value that has passed since 1970-01-01 09:00 and converts it to a datetime data type.

(TO_TIMESTAMP () converts a datetime data type to nanosecond data that has passed since 1970-01-01 09:00.)

```
FROM_TIMESTAMP(nanosecond_time_value)
```

```
Mach> SELECT FROM_TIMESTAMP(1562302560007248869) FROM TEST;
FROM_TIMESTAMP(1562302560007248869)
-----------------------------------
2019-07-05 13:56:00 007:248:869
```

Both sysdate and now represent nanosecond values elapsed since 1970-01-01 09:00 at the current time, so you can use FROM_TIMESTAMP () immediately.

Of course, the results are the same without using them. This can be useful if you have sysdate and now operations in nanoseconds.

```
Mach> select sysdate, from_timestamp(sysdate) from test_tbl;
sysdate                       from_timestamp(sysdate)
------------------------------------------------------------------
2019-07-05 14:00:59 722:822:443 2019-07-05 14:00:59 722:822:443
[1] row(s) selected.

Mach> select sysdate, from_timestamp(sysdate-1000000) from test_tbl;
sysdate                       from_timestamp(sysdate-1000000)
------------------------------------------------------------------
2019-07-05 14:01:05 130:939:525 2019-07-05 14:01:05 129:939:525      -- 1 ms (1,000,000 ns) difference
happens
[1] row(s) selected.
```

GROUP_CONCAT

This function is an aggregate function that outputs the value of the corresponding column in the group in a string.

```
GROUP_CONCAT(
     [DISTINCT] column
     [ORDER BY { unsigned_integer | column }
     [ASC | DESC] [, column ...]]
     [SEPARATOR str_val]
)
```

- DISTINCT: Duplicate values are not appended if duplicate values are attached.
- ORDER BY: Arranges the sequence of column values to be attached according to the specified column values.
- SEPARATOR: A delimiter string used to append column values. The default value is a comma (,).

The syntax notes are as follows.

- You can specify only one column, and if you want to specify more than one column, you must use the TO_CHAR () function and the CONCAT operator (||) to make one expression.
- ORDER BY can specify other columns besides the columns to be joined, and can specify multiple columns.
- You must enter a string constant in SEPARATOR, and you can not enter a string column.

```
Mach> CREATE TABLE concat_table(id1 INTEGER, id2 DOUBLE, name VARCHAR(10));
Created successfully.

Mach> INSERT INTO concat_table VALUES (1, 2, 'John');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (2, 1, 'Ram');
```

```
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (3, 2, 'Zara');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (4, 2, 'Jill');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (5, 1, 'Jack');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (6, 1, 'Jack');
1 row(s) inserted.


Mach> SELECT GROUP_CONCAT(name) AS G_NAMES FROM concat_table GROUP BY id2;
G_NAMES

--------------------------------------------------------------------------------
Jack,Jack,
Ram
Jill,Zara,
John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(DISTINCT name) AS G_NAMES FROM concat_table GROUP BY Id2;
G_NAMES

--------------------------------------------------------------------------------
Jack,
Ram
Jill,Zara,
John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(name SEPARATOR '.') G_NAMES FROM concat_table GROUP BY Id2;
G_NAMES

--------------------------------------------------------------------------------
Jack.Jack.
Ram
Jill.Zara.
John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(name ORDER BY id1) G_NAMES, GROUP_CONCAT(id1 ORDER BY id1) G_SORTID FROM
concat_table GROUP BY id2;
G_NAMES

--------------------------------------------------------------------------------
G_SORTID

--------------------------------------------------------------------------------
Ram,Jack,
Jack
2,5,6

John,Zara,
Jill
1,3,4

[2] row(s) selected.
```

INSTR

This function returns the index of the number of characters in the string entered together. The index starts at 1.

- If no string pattern is found, 0 is returned.

- If the length of the string pattern to find is 0 or NULL, NULL is returned.

```
INSTR(target_string, pattern_string)
```

```
Mach> CREATE TABLE string_table(c1 VARCHAR(20));
Created successfully.

Mach> INSERT INTO string_table VALUES ('abstract');
1 row(s) inserted.

Mach> INSERT INTO string_table VALUES ('override');
1 row(s) inserted.

Mach> SELECT c1, INSTR(c1, 'act') FROM string_table;
c1                   INSTR(c1, 'act')
-----------------------------------------
override             0
abstract             6
[2] row(s) selected
```

LEAST / GREATEST

Both functions return the smallest value (LEAST) or the largest value (GREATEST) if you specify multiple columns or values as input parameters.

If the input value is 1 or absent, it is treated as an error. If the input value is NULL, NULL is returned. Therefore, if the input value is a column, it must be converted in advance using a function.
If a column (BLOB, TEXT) that can not be compared with the input value is included or type conversion is not possible for comparison, comparison is processed as an error.

```
LEAST(value_list, value_list,...)
GREATEST(value_list, value_list,...)
```

```
Mach> CREATE TABLE lgtest_table(c1 INTEGER, c2 LONG, c3 VARCHAR(10), c4 VARCHAR(5));
Created successfully.

Mach> INSERT INTO lgtest_table VALUES (1, 2, 'abstract', 'ace');
1 row(s) inserted.

Mach> INSERT INTO lgtest_table VALUES (null, 100, null, 'bag');
1 row(s) inserted.

Mach> SELECT LEAST (c1, c2) FROM lgtest_table;
LEAST (c1, c2)
----------------------
NULL
1
[2] row(s) selected.

Mach> SELECT LEAST (c1, c2, -1) FROM lgtest_table;
LEAST (c1, c2, -1)
----------------------
NULL
-1
[2] row(s) selected.

Mach> SELECT GREATEST(c3, c4) FROM lgtest_table;
GREATEST(c3, c4)
--------------------
NULL
ace
[2] row(s) selected.

Mach> SELECT LEAST(c3, c4) FROM lgtest_table;
```

```
LEAST(c3, c4)
-----------------
NULL
abstract
[2] row(s) selected.

Mach> SELECT LEAST(NVL(c3, 'aa'), c4) FROM lgtest_table;
LEAST(NVL(c3, 'aa'), c4)
---------------------------
aa
abstract
[2] row(s) selected.
```

LENGTH

This function gets the length of a string column. The obtained value outputs the number of bytes in English.

```
LENGTH(column_name)
```

```
Mach> CREATE TABLE length_table (id1 INTEGER, id2 DOUBLE, name VARCHAR(15));
Created successfully.

Mach> INSERT INTO length_table VALUES(1, 10, 'Around the Horn');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(NULL, 20, 'Alfreds Futterkiste');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(3, NULL, 'Antonio Moreno');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(4, 40, NULL);
1 row(s) inserted.

Mach> select * FROM length_table;
ID1         ID2                         NAME
-------------------------------------------------------------
4           40                          NULL
3           NULL                        Antonio Moreno
NULL        20                          Alfreds Futterk
1           10                          Around the Horn
[4] row(s) selected.

Mach> select id1 * 10 FROM length_table;
id1 * 10
-----------------------
40
30
NULL
10
[4] row(s) selected.

Mach> select * FROM length_table Where id1 > 1 and id2 < 50;
ID1         ID2                         NAME
-------------------------------------------------------------
4           40                          NULL
[1] row(s) selected.

Mach> select name || ' with null concat' FROM length_table;
name || ' with null concat'
-----------------------------------
NULL
Antonio Moreno with null concat
Alfreds Futterk with null concat
```

```
Around the Horn with null concat
[4] row(s) selected.

Mach> select LENGTH(name) FROM length_table;
LENGTH(name)
---------------
NULL
14
15
15
[4] row(s) selected.
```

LOWER

This function converts an English string to lowercase.

```
LOWER(column_name)
```

```
Mach> CREATE TABLE lower_table (name VARCHAR(20));
Created successfully.

Mach> INSERT INTO lower_table VALUES('');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('James Backley');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('Alfreds Futterkiste');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('Antonio MORENO');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES (NULL);
1 row(s) inserted.

Mach> SELECT LOWER(name) FROM lower_table;
LOWER(name)
------------------------
NULL
antonio moreno
alfreds futterkiste
james backley
NULL
[5] row(s) selected.
```

LPAD / RPAD

This function adds a character to the left (LPAD) or to the right (RPAD) until the input is of a given length.

The last parameter, char, can be omitted, or a space `' '` character if omitted.
If the input column value is longer than the given length, the characters are not appended but only the length is taken from the beginning.

```
LPAD(str, len, padstr)
RPAD(str, len, padstr)
```

```
Mach> CREATE TABLE pad_table (c1 integer, c2 varchar(15));
Created successfully.
```

```
Mach> INSERT INTO pad_table VALUES (1, 'Antonio');
1 row(s) inserted.

Mach> INSERT INTO pad_table VALUES (25, 'Johnathan');
1 row(s) inserted.

Mach> INSERT INTO pad_table VALUES (30, 'M');
1 row(s) inserted.

Mach> SELECT LPAD(to_char(c1), 5, '0') FROM pad_table;
LPAD(to_char(c1), 5, '0')
-----------------------------
00030
00025
00001
[3] row(s) selected.

Mach> SELECT RPAD(to_char(c1), 5, '0') FROM pad_table;
RPAD(to_char(c1), 5, '0')
-----------------------------
30000
25000
10000
[3] row(s) selected.

Mach> SELECT LPAD(c2, 5) FROM pad_table;
LPAD(c2, 5)
---------------
    M
Johna
Anton
[3] row(s) selected.

Mach> SELECT RPAD(c2, 5) FROM pad_table;
RPAD(c2, 5)
---------------
M
Johna
Anton
[3] row(s) selected.

Mach> SELECT RPAD(c2, 10, '***') FROM pad_table;
RPAD(c2, 10, '***')
----------------------
M*********
Johnathan*
Antonio***
[3] row(s) selected.
```

LTRIM / RTRIM

This function removes the value corresponding to the pattern string from the first parameter. The LTRIM function checks to see if the characters are in pattern from left to right, the RTRIM function from right to left, and truncates until a character not in pattern is encountered. If all the strings are present in the pattern, NULL is returned.

If you do not specify a pattern expression, use the space character '' as a basis to remove the space character.

```
LTRIM(column_name, pattern)
RTRIM(column_name, pattern)
```

```
Mach> CREATE TABLE trim_table1(name VARCHAR(10));
Created successfully.

Mach> INSERT INTO trim_table1 VALUES ('   smith   ');
```

```
1 row(s) inserted.

Mach> SELECT ltrim(name) FROM trim_table1;
ltrim(name)
---------------
smith
[1] row(s) selected.

Mach> SELECT rtrim(name) FROM trim_table1;
rtrim(name)
---------------
   smith
[1] row(s) selected.

Mach> SELECT ltrim(name, ' s') FROM trim_table1;
ltrim(name, ' s')
---------------------
mith
[1] row(s) selected.

Mach> SELECT rtrim(name, 'h ') FROM trim_table1;
rtrim(name, 'h ')
---------------------
   smit
[1] row(s) selected.

Mach> CREATE TABLE trim_table2 (name VARCHAR(10));
Created successfully.

Mach> INSERT INTO trim_table2 VALUES ('ddckaaadkk');
1 row(s) inserted.

Mach> SELECT ltrim(name, 'dc') FROM trim_table2;
ltrim(name, 'dc')
---------------------
kaaadkk
[1] row(s) selected.

Mach> SELECT rtrim(name, 'dk') FROM trim_table2;
rtrim(name, 'dk')
---------------------
ddckaaa
[1] row(s) selected.

Mach> SELECT ltrim(name, 'dckak') FROM trim_table2;
ltrim(name, 'dckak')
------------------------
NULL
[1] row(s) selected.

Mach> SELECT rtrim(name, 'dckak') FROM trim_table2;
rtrim(name, 'dckak')
------------------------
NULL
[1] row(s) selected.
```

MAX

This function is an aggregate function that obtains the maximum value of a given numeric column.

```
MAX(column_name)
```

```
Mach> CREATE TABLE max_table (c INTEGER);
Created successfully.
```

```
Mach> INSERT INTO max_table VALUES(10);
1 row(s) inserted.

Mach> INSERT INTO max_table VALUES(20);
1 row(s) inserted.

Mach> INSERT INTO max_table VALUES(30);
1 row(s) inserted.

Mach> SELECT MAX(c) FROM max_table;
MAX(c)
--------------
30
[1] row(s) selected.
```

MIN

This function is an aggregate function that obtains the minimum value of a corresponding numeric column.

```
MIN(column_name)
```

```
Mach> CREATE TABLE min_table(c1 INTEGER);
Created successfully.

Mach> INSERT INTO min_table VALUES(1);
1 row(s) inserted.

Mach> INSERT INTO min_table VALUES(22);
1 row(s) inserted.

Mach> INSERT INTO min_table VALUES(33);
1 row(s) inserted.

Mach> SELECT MIN(c1) FROM min_table;
MIN(c1)
--------------
1
[1] row(s) selected.
```

NVL

This function returns value if the value of the column is NULL, or the value of the original column if it is not NULL.

```
NVL(string1, replace_with)
```

```
Mach> CREATE TABLE nvl_table (c1 varchar(10));
Created successfully.

Mach> INSERT INTO nvl_table VALUES ('Johnathan');
1 row(s) inserted.

Mach> INSERT INTO nvl_table VALUES (NULL);
1 row(s) inserted.

Mach> SELECT NVL(c1, 'Thomas') FROM nvl_table;
NVL(c1, 'Thomas')
```

```
--------------------
Thomas
Johnathan
```

ROUND

This function returns the result of rounding off the digits of the input value (input digit +1). If no digits are entered, the rounding is done at position 0. It is possible to enter a negative number in `decimals` place to round the decimal place.

```
ROUND(column_name, [decimals])
```

```
Mach> CREATE TABLE round_table (c1 DOUBLE);
Created successfully.

Mach> INSERT INTO round_table VALUES (1.994);
1 row(s) inserted.

Mach> INSERT INTO round_table VALUES (1.995);
1 row(s) inserted.

Mach> SELECT c1, ROUND(c1, 2) FROM round_table;
c1                          ROUND(c1, 2)
------------------------------------------------------------
1.995                       2
1.994                       1.99
```

ROWNUM

This function assigns a number to the SELECT query result row.

It can be used inside Subquery or Inline View that is used inside SELECT query. If you use ROWNUM () function in Inline View in Target List, you need to give Alias to refer to from outside.

```
ROWNUM()
```

Available Clauses

This function can be used in the target list, GROUP BY, or ORDER BY clause of a SELECT query. However, it can not be used in the WHERE and HAVING clauses of a SELECT query. ROWNUM () If you want to control WHERE or HAVING clause with result number, you can use SELECT query with ROWNUM () in Inline View and refer to it in WHERE or HAVING clause.

Available Clauses	Unavailable Clauses
Target List / GROUP BY / ORDER BY	WHERE / HAVING

```
Mach> CREATE TABLE rownum_table(c1 INTEGER, c2 DOUBLE, c3 VARCHAR(10));
Created successfully.

Mach> INSERT INTO rownum_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(2, 2.0, 'Second Row');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(3, 3.3, 'Third Row');
1 row(s) inserted.
```

```
Mach> INSERT INTO rownum_table VALUES(4, 4.3, 'Fourth Row');
1 row(s) inserted.

Mach> SELECT INNER_RANK, c3 AS NAME
    2 FROM   (SELECT ROWNUM() AS INNER_RANK, * FROM rownum_table)
    3 WHERE  INNER_RANK < 3;
INNER_RANK           NAME
-----------------------------------
1                    Fourth Row
2                    Third Row
[2] row(s) selected.
```

Altering Results Due to Sorting

If there is an ORDER BY clause in the SELECT query, the result number of ROWNUM () in the target list may not be sequentially assigned. This is because the ROWNUM () operation is performed before the operation of the ORDER BY clause. If you want to give it sequentially, you can use the query containing the ORDER BY clause in Inline View and then call ROWNUM () in the outer SELECT statement.

```
Mach> CREATE TABLE rownum_table(c1 INTEGER, c2 DOUBLE, c3 VARCHAR(10));
Created successfully.

Mach> INSERT INTO rownum_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(2, 2.0, 'John');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(3, 3.3, 'Sarah');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(4, 4.3, 'Micheal');
1 row(s) inserted.

Mach> SELECT ROWNUM(), c2 AS SORT, c3 AS NAME
    2 FROM   ( SELECT * FROM rownum_table ORDER BY c3 );
ROWNUM()             SORT                        NAME
-------------------------------------------------------------------
1                    1                           NULL
2                    2                           John
3                    4.3                         Micheal
4                    3.3                         Sarah
[4] row(s) selected.
```

SERIESNUM

Returns a number indicating how many of the records belong to the series grouped by SERIES BY. The return type is BIGINT type, and always returns 1 if the SERIES BY clause is not used.

```
SERIESNUM()
```

```
Mach> CREATE TABLE T1 (C1 INTEGER, C2 INTEGER);
Created successfully.

Mach> INSERT INTO T1 VALUES (0, 1);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (1, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (2, 3);
1 row(s) inserted.
```

```
Mach> INSERT INTO T1 VALUES (3, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (4, 1);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (5, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (6, 3);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (7, 1);
1 row(s) inserted.


Mach> SELECT SERIESNUM(), C1, C2 FROM T1 ORDER BY C1 SERIES BY C2 > 1;
SERIESNUM() C1 C2
-----------------------------------------------
1 1 2
1 2 3
1 3 2
2 5 2
2 6 3
[5] row(s) selected.
```

STDDEV / STDDEV_POP

This function is an aggregate function that returns the (standard) deviation and the population standard deviation of the (input) column. Equivalent to the square root of the VARIANCE and VAR_POP values, respectively.

```
STDDEV(column)
STDDEV_POP(column)
```

```
Mach> CREATE TABLE stddev_table(c1 INTEGER, C2 DOUBLE);

Mach> INSERT INTO stddev_table VALUES (1, 1);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (2, 1);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (3, 2);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (4, 2);
1 row(s) inserted.

Mach> SELECT c2, STDDEV(c1) FROM stddev_table GROUP BY c2;
c2                          STDDEV(c1)
------------------------------------------------------------
1                           0.707107
2                           0.707107
[2] row(s) selected.

Mach> SELECT c2, STDDEV_POP(c1) FROM stddev_table GROUP BY c2;
c2                          STDDEV_POP(c1)
------------------------------------------------------------
1                           0.5
2                           0.5
[2] row(s) selected.
```

SUBSTR

This function truncates the variable string column data from START to SIZE.

- START starts at 1 and returns NULL if it is zero.
- If SIZE is larger than the size of the corresponding string, only the maximum value of the string is returned.

SIZE is optional, and if omitted, it is internally specified by the size of the string.

```
SUBSTRING(column_name, start, [length])
```

```
Mach> CREATE TABLE substr_table (c1 VARCHAR(10));
Created successfully.

Mach> INSERT INTO substr_table values('ABCDEFG');
1 row(s) inserted.

Mach> INSERT INTO substr_table values('abstract');
1 row(s) inserted.

Mach> SELECT SUBSTR(c1, 1, 1) FROM substr_table;
SUBSTR(c1, 1, 1)
--------------------
a
A
[2] row(s) selected.

Mach> SELECT SUBSTR(c1, 3, 3) FROM substr_table;
SUBSTR(c1, 3, 3)
--------------------
str
CDE
[2] row(s) selected.

Mach> SELECT SUBSTR(c1, 2) FROM substr_table;
SUBSTR(c1, 2)
-----------------
bstract
BCDEFG
[2] row(s) selected.

Mach> drop table substr_table;
Dropped successfully.

Mach> CREATE TABLE substr_table (c1 VARCHAR(10));
Created successfully.

Mach> INSERT INTO substr_table values('ABCDEFG');
1 row(s) inserted.

Mach> SELECT SUBSTR(c1, 1, 1) FROM substr_table;
SUBSTR(c1, 1, 1)
--------------------
A
[1] row(s) selected.

Mach> SELECT SUBSTR(c1, 3, 3) FROM substr_table;
SUBSTR(c1, 3, 3)
--------------------
CDE
[1] row(s) selected.

Mach> SELECT SUBSTR(c1, 2) FROM substr_table;
SUBSTR(c1, 2)
-----------------
BCDEFG
[1] row(s) selected.
```

SUBSTRING_INDEX

Returns the duplicate string until the given delim is found by the count entered. If count is a negative value, it checks the delimiter from the end of the input string and returns it from the position where the delimiter was found to the end of the string.

If you enter count as 0 or there is no delimiter in the string, the function will return NULL.

```
SUBSTRING_INDEX(expression, delim, count)
```

```
Mach> CREATE TABLE substring_table (url VARCHAR(30));
Created successfully.

Mach> INSERT INTO substring_table VALUES('www.machbase.com');
1 row(s) inserted.

Mach> SELECT SUBSTRING_INDEX(url, '.', 1) FROM substring_table;
SUBSTRING_INDEX(url, '.', 1)
---------------------------------
www
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', 2) FROM substring_table;
SUBSTRING_INDEX(url, '.', 2)
---------------------------------
www.machbase
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', -1) FROM substring_table;
SUBSTRING_INDEX(url, '.', -1)
---------------------------------
com
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(SUBSTRING_INDEX(url, '.', 2), '.', -1) FROM substring_table;
SUBSTRING_INDEX(SUBSTRING_INDEX(url, '.', 2), '.', -1)
----------------------------------------
machbase
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', 0) FROM substring_table;
SUBSTRING_INDEX(url, '.', 0)
---------------------------------
NULL
[1] row(s) selected.
```

SUM

This function is an aggregate function that represents the sum of the numeric columns.

```
SUM(column_name)
```

```
Mach> CREATE TABLE sum_table (c1 INTEGER, c2 INTEGER);
Created successfully.

Mach> INSERT INTO sum_table VALUES(1, 1);
1 row(s) inserted.
```

```
Mach> INSERT INTO sum_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 3);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(3, 4);
1 row(s) inserted.

Mach> SELECT c1, SUM(c1) from sum_table group by c1;
c1          SUM(c1)
-----------------------------------
2           6
3           3
1           3
[3] row(s) selected.

Mach> SELECT c1, SUM(c2) from sum_table group by c1;
c1          SUM(c2)
-----------------------------------
2           6
3           4
1           6
[3] row(s) selected.
```

SYSDATE / NOW

SYSDATE is a pseudocolumn, not a function, that returns the system's current time.

NOW is the same function as SYSDATE and is provided for user convenience.

```
SYSDATE
NOW
```

```
Mach> SELECT SYSDATE, NOW FROM t1;

SYSDATE                         NOW
-------------------------------------------------------------------
2017-01-16 14:14:53 310:973:000 2017-01-16 14:14:53 310:973:000
```

TO_CHAR

This function converts a given data type to a string type. Depending on the type, format_string may be specified, but not for binary types.

```
TO_CHAR(column)
```

TO_CHAR: Default Datatype

The default data types are converted to data in the form of strings as shown below.

```
Mach> CREATE TABLE fixed_table (id1 SHORT, id2 INTEGER, id3 LONG, id4 FLOAT, id5 DOUBLE, id6 IPV4, id7 IPV6,
id8 VARCHAR (128));
Created successfully.

Mach> INSERT INTO fixed_table values(200, 19234, 1234123412, 3.14, 7.8338, '192.168.0.1', '::127.0.0.1',
'log varchar');
1 row(s) inserted.

Mach> SELECT '[ ' || TO_CHAR(id1) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id1) || ' ]'
--------------------------------------------------------------------------------
[ 200 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id2) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id2) || ' ]'
--------------------------------------------------------------------------------
[ 19234 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id3) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id3) || ' ]'
--------------------------------------------------------------------------------
[ 1234123412 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id4) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id4) || ' ]'
--------------------------------------------------------------------------------
[ 3.140000 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id5) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id5) || ' ]'
--------------------------------------------------------------------------------
[ 7.833800 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id6) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id6) || ' ]'
--------------------------------------------------------------------------------
[ 192.168.0.1 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id7) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id7) || ' ]'
--------------------------------------------------------------------------------
[ 0000:0000:0000:0000:0000:0000:7F00:0001 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id8) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id8) || ' ]'
--------------------------------------------------------------------------------
[ log varchar ]
[1] row(s) selected.
```

TO_CHAR : Floating point

Supported since 5.5.6.

This function converts the values of float and double columns into arbitrary strings.

Format expressions cannot be duplicated and must be entered in the form '[character] [number]'.

Format expression	Description

F / f	Specifies the number of decimal places for column values. The maximum input numeric value is 30.
N / n	Specify the number of decimal places for the column value, and enter a comma (,) every three digits for the integer part. The maximum input numeric value is 30

```
Mach> create table float_table (i1 float, i2 double);
Created successfully.

Mach> insert into float_table values (1.23456789, 1234.5678901234567890);
1 row(s) inserted.

Mach> select TO_CHAR(i1, 'f8'), TO_CHAR(i2, 'N9') from float_table;
TO_CHAR(i1, 'f8')       TO_CHAR(i2, 'N9')
------------------------------------------------------------
1.23456788              1,234.567890123
[1] row(s) selected.
```

TO_CHAR: DATETIME Type

A function that converts the value of a datetime column to an arbitrary string. You can use this function to create and combine various types of strings.

If format_string is omitted, the default is "YYYY-MM-DD HH24: MI: SS mmm: uuu: nnn".

Format Expression	Description
YYYY	Converts year to a four-digit number.
YY	Converts year to a two-digit number.
MM	Converts the month to a two-digit number.
MON	Converts the month to a three-digit abbreviated alphabet. (eg JAN, FEB, MAY, ...)
DD	Converts the day to a two-digit number.
DAY	Converts the day of the week to a three-digit abbreviation . (eg SUN, MON, ...)
IW	Converts the week number of a specific year from 1 to 53 (taking into account the day of the week) by the ISO 8601 rule . • The start of one week is Monday. • The first week can be considered as the last week of the previous year. Likewise, the last week can be considered the first week of the next year. See ISO 8601 more information .
WW	Converts week number of the particular year from 1 to 53 (Week Number) not taking into account the day of the week. That is, from January 1 to January 7, it is converted to 1.
W	Converts week number of a given month from 1 to 5 (Number The Week) not taking in to account the day of the week. That is, from March 1 to March 7 is converted to 1.
HH	Converts the time to a two-digit number.
HH12	Converts the time to a 2-digit number, from 1 to 12.
HH24	Converts the time to a 2-digit number, from 1 to 23.
HH2, HH3, HH6	Cuts the time to the number following HH. That is, when HH6 is used, 0 is expressed from 0 to 5, and 6 is expressed from 6 to 11. This expression is useful for calculating certain time-series statistics on time series. This value is expressed on a 24-hour basis.
MI	The minute is represented by a two-digit number.
MI2, MI5, MI10, MI20, MI30	The corresponding minute is cut to the number following MI. That is, when MI30 is used, 0 is expressed from 0 to 29 minutes, and 30 is represented from 30 to 59 minutes. This expression is useful for calculating certain time-series statistics on time series.
SS	The second is represented by a two-digit number.

SS2, SS5, SS10, SS20, SS30	The corresponding seconds are truncated to successive digits. That is, when SS30 is used, 0 is expressed from 0 to 29 seconds, and 30 is represented from 30 to 59 seconds. This expression is useful for calculating certain time-series statistics on time series.
AM	The current time is expressed in AM or PM according to AM and PM, respectively.
mmm	The millisecond of the time is represented by a three-digit number. The range of values is 0 to 999.
uuu	The micro second of the time is represented as a three-digit number. The range of values is 0 to 999.
nnn	The nano second of the time is expressed as a three-digit number. The range of values is 0 to 999.

```
Mach> CREATE TABLE datetime_table (id integer, dt datetime);
Created successfully.

Mach> INSERT INTO  datetime_table values(1, TO_DATE('1999-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(2, TO_DATE('2012-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(3, TO_DATE('2013-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(4, TO_DATE('2014-12-30 11:22:33 444:555:666'));
1 row(s) inserted.

Mach> SELECT id, dt FROM datetime_table WHERE dt > TO_DATE('2000-11-11 1:2:3 4:5:0');
id          dt
-----------------------------------------------
4           2014-12-30 11:22:33 444:555:666
3           2013-11-11 01:02:03 004:005:006
2           2012-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT id, dt FROM datetime_table WHERE dt > TO_DATE('2013-11-11 1:2:3') and dt < TO_DATE('2014-11-11
1:2:3');
id          dt
-----------------------------------------------
3           2013-11-11 01:02:03 004:005:006
[1] row(s) selected.

Mach> SELECT id, TO_CHAR(dt) FROM datetime_table;
id          TO_CHAR(dt)
-------------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33 444:555:666
3           2013-11-11 01:02:03 004:005:006
2           2012-11-11 01:02:03 004:005:006
1           1999-11-11 01:02:03 004:005:006
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY')
-------------------------------------------------------------------------------------------------
4           2014
3           2013
2           2012
1           1999
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM')
-------------------------------------------------------------------------------------------------
4           2014-12
3           2013-11
```

```
2           2012-11
1           1999-11
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD')
----------------------------------------------------------------------------------------------------
4           2014-12-30
3           2013-11-11
2           2012-11-11
1           1999-11-11
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD TO_CHAR') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD TO_CHAR')
----------------------------------------------------------------------------------------------------
4           2014-12-30 TO_CHAR
3           2013-11-11 TO_CHAR
2           2012-11-11 TO_CHAR
1           1999-11-11 TO_CHAR
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS')
----------------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33
3           2013-11-11 01:02:03
2           2012-11-11 01:02:03
1           1999-11-11 01:02:03
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS mmm.uuu.nnn') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS mmm.
----------------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33 444.555.666
3           2013-11-11 01:02:03 004.005.006
2           2012-11-11 01:02:03 004.005.006
1           1999-11-11 01:02:03 004.005.006
[4] row(s) selected.
```

TO_CHAR: Unsupported Type

Currently, TO_CHAR is not supported for binary types.

This is because it is impossible to convert to plain text. If you want to output it to the screen, you can check it by outputting hexadecimal value through TO_HEX () function.

TO_DATE

This function converts a string represented by a given format string to a datetime type.

If format_string is omitted, the default is "YYYY-MM-DD HH24: MI: SS mmm: uuu: nnn".

```
-- default format is "YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn" if no format exists.
TO_DATE(date_string [, format_string])
```

```
Mach> CREATE TABLE to_date_table (id INTEGER, dt datetime);
Created successfully.

Mach> INSERT INTO  to_date_table VALUES(1, TO_DATE('1999-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  to_date_table VALUES(2, TO_DATE('2012-11-11 1:2:3 4:5:6'));
1 row(s) inserted.
```

```
Mach> INSERT INTO  to_date_table VALUES(3, TO_DATE('2014-12-30 11:22:33 444:555:666'));
1 row(s) inserted.

Mach> INSERT INTO  to_date_table VALUES(4, TO_DATE('2014-12-30 23:22:34 777:888:999', 'YYYY-MM-DD HH24:MI:SS
mmm:uuu:nnn'));
1 row(s) inserted.

Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('1999-11-11 1:2:3 4:5:0');
id          dt
-------------------------------------------
4           2014-12-30 23:22:34 777:888:999
3           2014-12-30 11:22:33 444:555:666
2           2012-11-11 01:02:03 004:005:006
1           1999-11-11 01:02:03 004:005:006
[4] row(s) selected.

Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('2000-11-11 1:2:3 4:5:0');
id          dt
-------------------------------------------
4           2014-12-30 23:22:34 777:888:999
3           2014-12-30 11:22:33 444:555:666
2           2012-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('2012-11-11 1:2:3','YYYY-MM-DD HH24:MI:SS') and dt
< TO_DATE('2014-11-11 1:2:3','YYYY-MM-DD HH24:MI:SS');
id          dt
-------------------------------------------
2           2012-11-11 01:02:03 004:005:006
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999', 'YYYY') FROM to_date_table LIMIT 1;
id          TO_DATE('1999', 'YYYY')
-------------------------------------------
4           1999-01-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12', 'YYYY-MM') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12', 'YYYY-MM')
-------------------------------------------
4           1999.12.01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999', 'YYYY') FROM to_date_table LIMIT 1;
id          TO_DATE('1999', 'YYYY')
-------------------------------------------
4           1999-01-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12', 'YYYY-MM') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12', 'YYYY-MM')
-------------------------------------------
4           1999-12-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12', 'YYYY-MM-DD HH24:MI') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12', 'YYYY-MM-DD HH24:MI')
-----------------------------------------------------
4           1999-12-31 13:12:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32', 'YYYY-MM-DD HH24:MI:SS') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12:32', 'YYYY-MM-DD HH24:MI:SS')
-----------------------------------------------------
4           1999-12-31 13:12:32 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123', 'YYYY-MM-DD HH24:MI:SS mmm') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12:32 123', 'YYYY-MM-DD HH24:MI:SS mmm')
-----------------------------------------------------
```

```
4           1999-12-31 13:12:32 123:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123:456', 'YYYY-MM-DD HH24:MI:SS mmm:uuu') FROM to_date_table
LIMIT 1;
id          TO_DATE('1999-12-31 13:12:32 123:456', 'YYYY-MM-DD HH24:MI:SS mmm:uuu')
-----------------------------------------------------
4           1999-12-31 13:12:32 123:456:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123:456:789', 'YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn') FROM
to_date_table LIMIT 1;
id           TO_DATE('1999-12-31 13:12:32 123:456:789', 'YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn')
-----------------------------------------------------
4           1999-12-31 13:12:32 123:456:789
[1] row(s) selected.
```

TO_DATE_SAFE

Similar to TO_DATE (), but returns NULL without error if conversion fails.

```
TO_DATE_SAFE(date_string [, format_string])
```

```
Mach> CREATE TABLE date_table (ts DATETIME);
Created successfully.

Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-01-01', 'YYYY-MM-DD'));
1 row(s) inserted.
Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-01-02', 'YYYY'));
1 row(s) inserted.
Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-12-32', 'YYYY-MM-DD'));
1 row(s) inserted.

Mach> SELECT ts FROM date_table;
ts
---------------------------------
NULL
NULL
2016-01-01 00:00:00 000:000:000
[3] row(s) selected.
```

TO_HEX

This function returns value if the value of the column is NULL, or the value of the original column if it is not NULL. To ensure consistency of output, convert to BIG ENDIAN type for short, int, and long types.

```
TO_HEX(column)
```

```
Mach> CREATE TABLE hex_table (id1 SHORT, id2 INTEGER, id3 VARCHAR(10), id4 FLOAT, id5 DOUBLE, id6 LONG, id7
IPV4, id8 IPV6, id9 TEXT, id10 BINARY,
id11 DATETIME);
Created successfully.

Mach> INSERT INTO hex_table VALUES(256, 65535, '0123456789', 3.141592, 1024 * 1024 * 1024 * 3.14,
13513135446, '192.168.0.1', '::192.168.0.1', 'textext',
'binary', TO_DATE('1999', 'YYYY'));
1 row(s) inserted.
```

```
Mach> SELECT TO_HEX(id1), TO_HEX(id2), TO_HEX(id3), TO_HEX(id4), TO_HEX(id5), TO_HEX(id6), TO_HEX(id7),
TO_HEX(id8), TO_HEX(id9), TO_HEX(id10), TO_HEX(id11)
FROM hex_table;
TO_HEX(id1)  TO_HEX(id2)  TO_HEX(id3)            TO_HEX(id4)  TO_HEX(id5)        TO_HEX(id6)        TO_HEX
(id7)
-------------------------------------------------------------------------------------------------------------------
------------
TO_HEX(id8)                          TO_HEX(id9)
-------------------------------------------------------------------------------------------------------------------
-------------
TO_HEX(id10)                                                                          TO_HEX(id11)
-------------------------------------------------------------------------------------------------------------------
0100   0000FFFF   30313233343536373839   D80F4940   1F85EB51B81EE941   0000000325721556   04C0A80001
0600000000000000000000000C0A80001   74657874657874
62696E617279                                                                          0CB325846E226000
[1] row(s) selected.
```

TO_IPV4 / TO_IPV4_SAFE

This function converts a given string to an IP version 4 type. If the string can not be converted to a numeric value, the TO_IPV4 () function returns an error and terminates the operation.

However, in the case of the TO_IPV4_SAFE () function, NULL is returned in case of error, and the operation can continue.

```
TO_IPV4(string_value)
TO_IPV4_SAFE(string_value)
```

```
Mach> CREATE TABLE ipv4_table (c1 varchar(100));
Created successfully.

Mach> INSERT INTO ipv4_table VALUES('192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipv4_table VALUES('     192.168.0.2    ');
1 row(s) inserted.

Mach> INSERT INTO ipv4_table VALUES(NULL);
1 row(s) inserted.

Mach> SELECT c1 FROM ipv4_table;
c1
-------------------------------------------------------------------------------
NULL
     192.168.0.2
192.168.0.1
[3] row(s) selected.

Mach> SELECT TO_IPV4(c1) FROM ipv4_table;
TO_IPV4(c1)
------------------
NULL
192.168.0.2
192.168.0.1
[3] row(s) selected.

Mach> INSERT INTO ipv4_table VALUES('192.168.0.1.1');
1 row(s) inserted.

Mach> SELECT TO_IPV4(c1) FROM ipv4_table limit 1;
TO_IPV4(c1)
------------------
[ERR-02068 : Invalid IPv4 address format (192.168.0.1.1).]
[0] row(s) selected.

Mach> SELECT TO_IPV4_SAFE(c1) FROM ipv4_table;
```

```
TO_IPV4_SAFE(c1)
-------------------
NULL
NULL
192.168.0.2
192.168.0.1
[4] row(s) selected.
```

TO_IPV6 / TO_IPV6_SAFE

This function converts a given string to an IP version 6 type. If the string can not be converted to a numeric type, the TO_IPV6 () function returns an error and terminates the operation.

However, in the case of the TO_IPV6_SAFE () function, NULL is returned in case of error, and the operation can continue.

```
TO_IPV6(string_value)
TO_IPV6_SAFE(string_value)
```

```
Mach> CREATE TABLE ipv6_table (id varchar(100));
Created successfully.

Mach> INSERT INTO ipv6_table VALUES('::0.0.0.0');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0' || '.0.2');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('   ::127.0.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0.0.4  ');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('   ::FFFF:255.255.255.255   ');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('21DA:D3:0:2F3B:2AA:FF:FE28:9C5A');
1 row(s) inserted.

Mach> SELECT TO_IPV6(id) FROM ipv6_table;
TO_IPV6(id)
----------------------------------------------------------------
21da:d3::2f3b:2aa:ff:fe28:9c5a
::ffff:255.255.255.255
::127.0.0.4
::127.0.0.3
::127.0.0.2
::127.0.0.1
::
[7] row(s) selected.

Mach> INSERT INTO ipv6_table VALUES('127.0.0.10.10');
1 row(s) inserted.

Mach> SELECT TO_IPV6(id) FROM ipv6_table limit 1;
TO_IPV6(id)
----------------------------------------------------------------
[ERR-02148 : Invalid IPv6 address format.(127.0.0.10.10)]
[0] row(s) selected.

Mach> SELECT TO_IPV6_SAFE(id) FROM ipv6_table;
```

```
TO_IPV6_SAFE(id)
------------------------------------------------------------
NULL
21da:d3::2f3b:2aa:ff:fe28:9c5a
::ffff:255.255.255.255
::127.0.0.4
::127.0.0.3
::127.0.0.2
::127.0.0.1
::
[8] row(s) selected.
```

TO_NUMBER / TO_NUMBER_SAFE

This function converts a given string to a numeric double. If the string can not be converted to a numeric value, the TO_NUMBER () function returns an error and terminates the operation.

However, in case of TO_NUMBER_SAFE () function, NULL is returned in case of error, and the operation can continue.

```
TO_NUMBER(string_value)
TO_NUMBER_SAFE(string_value)
```

```
Mach> CREATE TABLE number_table (id varchar(100));
Created successfully.

Mach> INSERT INTO number_table VALUES('10');
1 row(s) inserted.

Mach> INSERT INTO number_table VALUES('20');
1 row(s) inserted.

Mach> INSERT INTO number_table VALUES('30');
1 row(s) inserted.

Mach> SELECT TO_NUMBER(id) from number_table;
TO_NUMBER(id)
------------------------------
30
20
10
[3] row(s) selected.

Mach> CREATE TABLE safe_table (id varchar(100));
Created successfully.

Mach> INSERT INTO safe_table VALUES('invalidnumber');
1 row(s) inserted.

Mach> SELECT TO_NUMBER(id) from safe_table;
TO_NUMBER(id)
------------------------------
[ERR-02145 : The string cannot be converted to number value.(invalidnumber)]
[0] row(s) selected.

Mach> SELECT TO_NUMBER_SAFE(id) from safe_table;
TO_NUMBER_SAFE(id)
------------------------------
NULL
[1] row(s) selected.
```

TO_TIMESTAMP

This function converts a datetime data type to nanosecond data that has passed since 1970-01-01 09:00.

```
TO_TIMESTAMP(datetime_value)
```

```
Mach> create table datetime_tbl (c1 datetime);
Created successfully.

Mach> insert into datetime_tbl values ('2010-01-01 10:10:10');
1 row(s) inserted.

Mach> select to_timestamp(c1) from datetime_tbl;
to_timestamp(c1)
----------------------
1262308210000000000
[1] row(s) selected.
```

TRUNC

The TRUNC function returns the number truncated at the nth place after the decimal point.

If n is omitted, treat it as 0 and delete all decimal places. If n is negative, it returns the value truncated from n before the decimal point.

```
TRUNC(number [, n])
```

```
Mach> CREATE TABLE trunc_table (i1 DOUBLE);
Created successfully.

Mach> INSERT INTO trunc_table VALUES (158.799);
1 row(s) inserted.

Mach> SELECT TRUNC(i1, 1), TRUNC(i1, -1) FROM trunc_table;
TRUNC(i1, 1)                TRUNC(i1, -1)
------------------------------------------------------------
158.7                       150
[1] row(s) selected.

Mach> SELECT TRUNC(i1, 2), TRUNC(i1, -2) FROM trunc_table;
TRUNC(i1, 2)                TRUNC(i1, -2)
------------------------------------------------------------
158.79                      100
[1] row(s) selected.
```

TS_CHANGE_COUNT

This function is an aggregate function that obtains the number of changes to a particular column value.

This function can not be used with 1) Join or 2) Inline view because it can be guaranteed that input data is input in chronological order.
The current version only supports types except varchar.

```
TS_CHANGE_COUNT(column)
```

```
Mach> CREATE TABLE ipcount_table (id INTEGER, ip IPV4);
Created successfully.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.2');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.2');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.4');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.4');
1 row(s) inserted.

Mach> SELECT id, TS_CHANGE_COUNT(ip) from ipcount_table GROUP BY id;
id          TS_CHANGE_COUNT(ip)
------------------------------------
2           2
1           4
[2] row(s) selected.
```

UNIX_TIMESTAMP

UNIX_TIMESTAMP is a function that converts a date type value to a 32-bit integer value that is converted by unix's time () system call. (FROM_UNIXTIME is a function that converts integer data to a date type value on the contrary.)

```
UNIX_TIMESTAMP(datetime_value)
```

```
Mach> CREATE table unix_table (c1 int);
Created successfully.

Mach> INSERT INTO unix_table VALUES (UNIX_TIMESTAMP('2001-01-01'));
1 row(s) inserted.

Mach> SELECT * FROM unix_table;
C1
--------------
978274800
[1] row(s) selected.
```

UPPER

This function converts the contents of a given English column to uppercase.

```
UPPER(string_value)
```

```
Mach> CREATE TABLE upper_table(id INTEGER,name VARCHAR(10));
Created successfully.

Mach> INSERT INTO upper_table VALUES(1, '');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(2, 'James');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(3, 'sarah');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(4, 'THOMAS');
1 row(s) inserted.

Mach> SELECT id, UPPER(name) FROM upper_table;
id          UPPER(name)
----------------------------
4           THOMAS
3           SARAH
2           JAMES
1           NULL
[4] row(s) selected.
```

VARIANCE / VAR_POP

This function is an aggregate function that returns the variance of a given numeric column value. The Variance function returns the variance for the sample, and the VAR_POP function returns the variance for the population.

```
VARIANCE(column_name)
VAR_POP(column_name)
```

```
Mach> CREATE TABLE var_table(c1 INTEGER, c2 DOUBLE);
Created successfully.

Mach> INSERT INTO var_table VALUES (1, 1);
1 row(s) inserted.

Mach> INSERT INTO var_table VALUES (2, 1);
1 row(s) inserted.

Mach> INSERT INTO var_table VALUES (1, 2);
1 row(s) inserted.

Mach> INSERT INTO var_table VALUES (2, 2);
1 row(s) inserted.

Mach> SELECT VARIANCE(c1) FROM var_table;
VARIANCE(c1)
------------------------------
0.333333
[1] row(s) selected.

Mach> SELECT VAR_POP(c1) FROM var_table;
VAR_POP(c1)
------------------------------
0.25
[1] row(s) selected.
```

YEAR / MONTH / DAY

These functions extract the corresponding year, month, and day from the input datetime column value and return it as an integer type value.

```
YEAR(datetime_col)
MONTH(datetime_col)
DAY(datetime_col)
```

```
Mach> CREATE TABLE extract_table(c1 DATETIME, c2 INTEGER);
Created successfully.

Mach> INSERT INTO extract_table VALUES (to_date('2001-01-01 12:30:00 000:000:000'), 1);
1 row(s) inserted.

Mach> SELECT YEAR(c1), MONTH(c1), DAY(c1) FROM extract_table;
year(c1)    month(c1)   day(c1)
---------------------------------------
2001        1           1
```

Support Type of Built-In Function

	Short	Integer	Long	Float	Double	Varchar	Text	Ipv4	Ipv6	Datetime	Binary
ABS	o	o	o	o	o	x	x	x	x	x	x
ADD_TIME	x	x	x	x	x	x	x	x	x	o	x
AVG	o	o	o	o	o	x	x	x	x	x	x
BITAND / BITOR	o	o	o	x	x	x	x	x	x	x	x
COUNT	o	o	o	o	o	o	x	o	o	o	x
DATE_TRUNC	x	x	x	x	x	x	x	x	x	o	x
DECODE	o	o	o	o	o	o	x	o	x	o	x
FIRST / LAST	o	o	o	o	o	o	x	o	o	o	x
FROM_UNIXTIME	o	o	o	o	o	x	x	x	x	x	x
FROM_TIMESTAMP	o	o	o	o	o	x	x	x	x	x	x
GROUP_CONCAT	o	o	o	o	o	o	x	o	o	o	x
INSTR	x	x	x	x	x	o	o	x	x	x	x
LEAST / GREATEST	o	o	o	o	o	o	x	x	x	x	x
LENGTH	x	x	x	x	x	o	o	x	x	x	o
LOWER	x	x	x	x	x	o	x	x	x	x	x
LPAD / RPAD	x	x	x	x	x	o	x	x	x	x	x
LTRIM / RTRIM	x	x	x	x	x	o	x	x	x	x	x
MAX	o	o	o	o	o	o	x	o	o	o	x
MIX	o	o	o	o	o	o	x	o	o	o	x
NVL	x	x	x	x	x	o	x	o	x	x	x
ROUND	o	o	o	o	o	x	x	x	x	x	x
	o	o	o	o	o	o	o	o	o	o	o

ROWNUM											
SERIESNUM	o	o	o	o	o	o	o	o	o	o	o
STDDEV / STDDEV_POP	o	o	o	o	o	x	x	x	x	x	x
SUBSTR	x	x	x	x	x	o	x	x	x	x	x
SUBSTRING_INDEX	x	x	x	x	x	o	o	x	x	x	x
SUM	o	o	o	o	o	x	x	x	x	x	x
SYSDATE / NOW	x	x	x	x	x	x	x	x	x	x	x
TO_CHAR	o	o	o	o	o	o	x	o	o	o	x
TO_DATE / TO_DATE_SAFE	x	x	x	x	x	o	x	x	x	x	x
TO_HEX	o	o	o	o	o	o	o	o	o	o	o
TO_IPV4 / TO_IPV4_SAFE	x	x	x	x	x	o	x	x	x	x	x
TO_IPV6 / TO_IPV6_SAFE	x	x	x	x	x	o	x	x	x	x	x
TO_NUMBER / TO_NUMBER_SAFE	x	x	x	x	x	o	x	x	x	x	x
TO_TIMESTAMP	x	x	x	x	x	x	x	x	x	o	x
TRUNC	o	o	o	o	o	x	x	x	x	x	x
TS_CHANGE_COUNT	o	o	o	o	o	x	x	o	o	o	x
UNIX_TIMESTAMP	o	o	o	o	o	x	x	x	x	x	x
UPPER	x	x	x	x	x	o	x	x	x	x	x
VARIANCE / VAR_POP	o	o	o	o	o	x	x	x	x	x	x
YEAR / MONTH / DAY	x	x	x	x	x	x	x	x	x	o	x