SELECT

SELECT is a syntax used to find, filter, and manipulate data from various tables in Machbase.

SELECT Syntax

```
select_stmt UNION ALL select_stmt
```

```
SELECT target_list FROM TableList WHERE Condition GROUP BY Expr ORDER BY
Expr [Desc] HAVING Expr SERIES BY Expr LIMIT N[,N] DURATION TimeExpr;
```

Index

SET OPERATOR

Used when receiving the results of multiple Select queries as a single query result. Machbase supports only the UNION ALL set operator. The set operator can be executed only if the left and right Select statements are (1) the same or compatible types, (2) the number of query results is the same, and if any of the two conditions does not match, they are treated as errors.

Data type conversion and compatibility verification are performed based on the following criteria.

- Signed integer types and unsigned integer types are not compatible.
- The integer type is compatible with the real type, and the query result is converted to the real type and returned.
- Character types are compatible with different lengths.
- IPv6 type and IPv4 type are not compatible.
- Of the two SELECT statements, the column name of the left query is always used.

Examples

```
SELECT i1, i2 FROM table_1
UNION ALL
SELECT c1, c2 FROM table_2
```

    - NOT SEARCH Statement
    - REGEXP Statement
    - Use In Statement and SUBQUERY
    - BETWEEN Statement
    - RANGE Statement
- GROUP BY / HAVING
- ORDER BY
- SERIES BY
- LIMIT
- DURATION
- SAVE DATA

TARGET LIST

This is a **list of columns or subqueries** targeted by the Select statement .

The subquery used in the target list is treated as an error if it has two or more values or two or more result columns, such as a subquery used in the WHERE clause.

```
SELECT i1, i2 ...
SELECT i1 (Select avg(c1) FROM t1), i2 ...
```

CASE Statement

```
CASE <simple_case_expression|searched_case_expression> [else_clause] END

simple_case_expression ::=
    expr WHEN comparison_expr THEN return_expr
        [WHEN comparison_expr THEN return_expr ...]

searched_case_expression ::=
    WHEN condtion_expr THEN return EXPR [WHEN condtion_expr THEN return EXPR ...]

else_clause ::=
    ELSE else_value_expr
```

This is an expression that supports the IF ... THEN ... ELSE block of a typical programming language. simple_case_expression is executed in the form of return_expr when one column or expression is equal to the value of comparison_expr followed by when, and this when ... then clause can be repeated as many times as desired.

searched_case_expression does not specify an expression after CASE but describes a conditional clause that includes a comparison operator in the when clause. If the result of each comparison operation is true, then the value of the then clause is returned. The else clause returns else_value if the value of the when clause is not satisfied (even if the expression is NULL).

```
select * from t1;
I1          I2
---------------------------
2           2
1           1
[2] row(s) selected.

select case i1 when 1 then 100 end from t1;
case i1 when 1 then 100 end
------------------------------
NULL
100
[2] row(s) selected.
```

In the simple_case_expression example, if the value of the i1 column is 2, NULL is returned.

```
select case when i1 > 0 then 100 when i1 > 1 then 200 end from t1;
case when i1 > 0 then 100 when i1 > 1 then 200 end
------------------------------------------
100
100
[2] row(s) selected.
```

Since searched_case_expression returns the first condition that satisfies the condition, 100 is returned, and the second condition is not executed.

FROM

You can specify a table name or an Inline view in the FROM clause. To perform a join between tables, lists the table or Inline view separated by a comma (,).

```
FROM table_name
```

Retrieves data in the table specified by table_name.

SUBQUERY(INLINE VIEW)

```
FROM (Select statement)
```

Retrievse data for the contents of the subquery enclosed in parentheses.

> Machbase server does not support correlated subqueries, so you can not reference columns in a subquery in an outer query.

JOIN(INNER JOIN)

```
FROM TABLE_1, TABLE_2
```

Joins two tables, table_1 and table_2. An INNER JOIN can be used when three or more tables are listed, and both the search condition and the conditional clause are described in the WHERE clause.

```
SELECT t1.i1, t2.i1 FROM t1, t2 WHERE t1.i1 = t2.i1 AND t1.i1 > 1 AND t2.i2 = 3;
```

INNER JOIN OUTER JOIN

Supports ANSI style INNER JOIN, LEFT OUTER JOIN, and RIGHT OUTER JOIN. FULL OUTER JOIN is not supported.

```
FROM TABLE_1 [INNER|LEFT OUTER|RIGHT OUTER] JOIN TABLE_2 ON expression
```

The ON clause of the ANSI-style JOIN clause uses the conditional clause that is performed by the JOIN. If the WHERE clause in the OUTER JOIN query has a clause for an inner table (a table that is filled with NULL if the condition of the ON clause is not satisfied), the query is converted to an INNER JOIN.

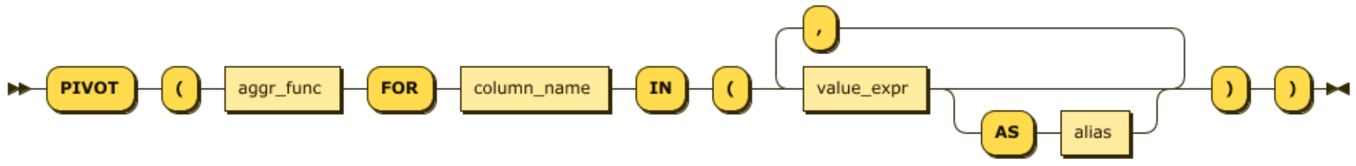
```
SELECT t1.i1 t2.i1 FROM t1 LEFT OUTER JOIN t2 ON (t1.i1 = t2.i1) WHERE t2.i2 = 1;
```

The above query is converted to an INNER JOIN by the condition t2.i2 = 1 in the WHERE clause.

PIVOT

> The PIVOT syntax is supported from Machbase version 5.6.

pivot_clause:



The PIVOT statement shows the aggregated results of GROUP BY output as ROW, rearranged into columns.

It is used in conjunction with the Inline view and is performed as follows.

- Performs GROUP BY on columns that are not used in the PIVOT clause of the inline view, and then performs aggregate functions on the values listed in the PIVOT IN clause.
- The resulting grouping column and the aggregation result are rotated and displayed as columns.

For example, aggregate the value of each device from the data collected from various sensors.
The query that should be performed through the CASE statement can be expressed simply through the PIVOT statement.

```
-- w/o PIVOT
SELECT * FROM (
    SELECT
             regtime,
             SUM(CASE WHEN tagid = 'FRONT_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS front_axis_torque,
             SUM(CASE WHEN tagid = 'REAR_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS rear_axis_torque,
             SUM(CASE WHEN tagid = 'HOIST_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS hoist_axis_torque,
             SUM(CASE WHEN tagid = 'SLIDE_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS slide_axis_torque
    FROM     result_d
    WHERE    regtime BETWEEN TO_DATE('2018-12-07 00:00:00') AND TO_DATE('2018-12-08 05:00:00')
    GROUP BY regtime
) WHERE front_axis_torque >= 40 AND rear_axis_torque >= 20;

-- w/ PIVOT
SELECT * FROM (
    SELECT regtime, tagid, dvalue FROM result_d
    WHERE  regtime BETWEEN TO_DATE('2018-12-07 00:00:00') AND TO_DATE('2018-12-08 05:00:00')
) PIVOT (SUM(dvalue) FOR tagid IN ('FRONT_AXIS_TORQUE', 'REAR_AXIS_TORQUE', 'HOIST_AXIS_TORQUE',
'SLIDE_AXIS_TORQUE'))
WHERE front_axis_torque >= 40 AND rear_axis_torque >= 20;

-- Result
regtime                         'FRONT_AXIS_TORQUE'         'REAR_AXIS_TORQUE'
'HOIST_AXIS_TORQUE'          'SLIDE_AXIS_TORQUE'
--------------------------------------------------------------------------------------------------------------
-----------------------------------------
2018-12-07 16:42:29 840:000:000 12158                      7244
NULL                         NULL
2018-12-07 14:56:26 220:000:000 3308                       663
NULL                         NULL
2018-12-07 12:20:13 844:000:000 3804                       113
NULL                         NULL
2018-12-07 11:10:01 957:000:000 8729                       5384
NULL                         NULL
2018-12-07 17:46:57 812:000:000 7500                       4559
NULL                         NULL
2018-12-07 14:30:06 138:000:000 5080                       6817
NULL                         -429
2018-12-07 13:09:20 464:000:000 5233                       1869
-7253                        NULL
2018-12-07 15:43:03 539:000:000 7491                       4453
NULL                         NULL
...
```

WHERE

Use of SUBQUERY

Subquery can be used for conditional statements. If the subquery returns more than one record in a clause except the IN clause, or if there is more than one result column in the subquery, it is not supported.

```
WHERE i1 = (SELECT MAX(c2) FROM T1)
```

Uses subquery by surrounding parentheses to the right of the conditional operator.

Machbase server does not support correlated subqueries, so you can not reference columns in a subquery in an outer query.

SEARCH Statement

The syntax is the same as for a regular database. However, a keyword index must be registered, and an additional search operation is possible by adding "SEARCH" as an operator keyword for text search.

```
-- drop table realdual;
create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));

create keyword index idx1 on realdual (id2);
create keyword index idx2 on realdual (id3);

insert into realdual values(1, 'time time2', 'series series2');

select * from realdual;

select * from realdual where id2 search 'time';
select * from realdual where id3 search 'series' ;
select * from realdual where id2 search 'time' and id3 search 'series';
```

The results are as follows.

```
Mach> create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));
Created successfully.

Mach> create keyword index idx1 on realdual (id2);
Created successfully.

Mach> create keyword index idx2 on realdual (id3);
Created successfully.

Mach> insert into realdual values(1, 'time time2', 'series series2');
1 row(s) inserted.

Mach> select * from realdual;
ID1         ID2                   ID3
------------------------------------------------------------
1           time time2            series series2
[1] row(s) selected.

Mach> select * from realdual where id2 search 'time';
ID1         ID2                   ID3
------------------------------------------------------------
1           time time2            series series2
[1] row(s) selected.

Mach> select * from realdual where id3 search 'series';
ID1         ID2                   ID3
------------------------------------------------------------
1           time time2            series series2
[1] row(s) selected.

Mach> select * from realdual where id2 search 'time' and id3 search 'series';
ID1         ID2                   ID3
------------------------------------------------------------
1           time time2            series series2
[1] row(s) selected.
```

ESEARCH Statement

The ESEARCH statement is a search keyword that enables extended searches on ASCII text. For this extension, search for the desired pattern is performed using the % character. In this Like operation, if all the records are checked before the %, the advantage of ESEARCH is that the words can be found quickly even in this case. This feature can be very useful when looking for a part of an English string (an error string or code).

```
Example

select id2 from realdual where id2 esearch 'bbb%';
id2
--------------------------------------------
bbb ccc1
aaa bbb1

[2] row(s) selected.

Search pattern 'bbb%' also includes bbb1 in search results.


select id3 from realdual where id3 esearch '%cd%';
id3
--------------------------------------------
cdf def1
bcd/cdf1ad
abc, bcd1
[3] row(s) selected.

% character works in middle of search pattern as well as beginning and end.

select id3 from realdual where id3 esearch '%cd%';
id3
--------------------------------------------
cdf def1
bcd/cdf1ad
abc, bcd1
[3] row(s) selected.
```

NOT SEARCH Statement

NOT SEARCH is a statement that returns true for records other than those found in the SEARCH statement.

NOT ESEARCH can not be used.

```
create table t1 (id integer, i2 varchar(10));
create keyword index t1_i2 on t1(i2);
insert into t1 values (1, 'aaaa');
insert into t1 values (2, 'bbbb');

select id from t1 where i2 not search 'aaaa';

id
--------------------------------------------
2
[1] row(s) selected.
```

REGEXP Statement

The REGEXP statement is used to perform searches on data using regular expressions. In general, patterns of a particular column are filtered using regular expressions.

One thing to keep in mind is that you can not use indexes when using the REGEXP clause, so you must lower the overall search cost by putting index conditions on other columns in order to reduce the overall search scope.
If you want to check a specific pattern, use index by SEARCH or ESEARCH, and then use REGEXP again in a state where the total number of data is small, it helps to improve the efficiency of the whole system.

```
Mach>
create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));
create table dual (id integer);
insert into dual values(1);
insert into realdual values(1, 'time1', 'series1 series21');
insert into realdual values(1, 'time2', 'series2 series22');
insert into realdual values(1, 'time3', 'series3 series32');


Mach> select * from realdual where id2 REGEXP 'time' ;
ID1         ID2                   ID3
------------------------------------------------------------
1           time3                 series3 series32
1           time2                 series2 series22
1           time1                 series1 series21
[3] row(s) selected.

Mach> select * from realdual where id2 REGEXP 'time[12]' ;
ID1         ID2                   ID3
------------------------------------------------------------
1           time2                 series2 series22
1           time1                 series1 series21
[2] row(s) selected.

Mach> select * from realdual where id2 REGEXP 'time[13]' ;
ID1         ID2                   ID3
------------------------------------------------------------
1           time3                 series3 series32
1           time1                 series1 series21
[2] row(s) selected.

Mach> select * from realdual where id2 regexp 'time[13]' and id3 regexp 'series[12]';
ID1         ID2                   ID3
------------------------------------------------------------
1           time1                 series1 series21
[1] row(s) selected.

Mach> select * from realdual where id2 NOT REGEXP 'time[12]';
ID1         ID2                   ID3
------------------------------------------------------------
1           time3                 series3 series32
[1] row(s) selected.

Mach> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e' from dual;
'abcde' REGEXP 'a[bcd]{1,10}e'
---------------------------------
1
[1] row(s) selected.
```

IN Statement

```
column_name IN (value1, value2,...)
```

The IN statement returns TRUE if it is satisfied in the value list. It is the same as the syntax linked by OR.

Use In Statement and SUBQUERY

You can use a subquery to the right of the IN statement in the conditional statement. However, if you specify more than one column on the left side of the IN condition, it treats it as an error and checks whether the result set returned from the right subquery exists in the left column value.

```
WHERE i1 IN (Select c1 from ...)
```

Machbase server does not support correlated subqueries, so you can not reference columns in a subquery in an outer query.

BETWEEN Statement

```
column_name BETWEEN value1 AND value2
```

The BETWEEN statement returns TRUE if the value of column is in the range of value1 and value2.

RANGE Statement

```
column_name RANGE duration_spec;

-- duration_spec : integer (YEAR | WEEK | HOUR | MINUTE | SECOND);
```

Provides a Range operator that allows you to easily specify a time condition for a given column. The Range operator specifies the time range from the current time as the target of the operation, rather than specifying a specific time (as specified by the BEFORE keyword). With this operator, you can easily retrieve result records within a desired time range.

```
select * from test where id < 2 and c1 range 1 hour;
ID          C1
-----------------------------------------------
1           2014-07-25 09:28:53 706:707:001
[1] row(s) selected.
```

GROUP BY / HAVING

The GROUP BY clause is used to group the results of a SELECT statement on a specific column. It is used when sorting by group or by aggregating functions by using aggregate functions. Group means records having the same column value for the column specified in the GROUP BY clause.You can combine the HAVING clause after the GROUP BY clause to set the conditional expression for group selection. That is, of all the groups constituted by the GROUP BY clause, only the group satisfying the conditional expression specified in the HAVING clause is inquired.

```
SELECT ...
GROUP BY { col_name | expr } ,...[ HAVING <search_condition> ]

select id1, avg(id2) from exptab where id2 group by id1 order by id1;
Obtain average value of id2 based on id1 column.
```

ORDER BY

The ORDER BY clause sorts the query results in ascending or descending order. If no sorting options such as ASC or DESC are specified, the ORDER BY clause sorts by default in ascending order. If the ORDER BY clause is not specified, the order of the records to be queried depends on the query.

```
SELECT ...
ORDER BY {col_name | expr} [ASC | DESC]

select id1, avg(id2) from exptab where id2 group by id1 order by id1;
Obtain average value of id2 based on id1 column.
```

SERIES BY

The SERIES BY clause extracts the sorted result set as successive result values satisfying the SERIES BY condition. If the ORDER BY clause is not specified, it generates the sorted result using the _ARRIVAL_TIME column value. Therefore, if you use the GROUP BY clause or the query for a volatile table or lookup table that does not have the _ARRIVAL_TIME column, you must use the ORDER BY clause do.

The result values that satisfy the conditional clause will have the return value of the same SERIESNUM () function.

```
For example, for the following data
CREATE TABLE T1 (C1 INTEGER, C2 INTEGER);
INSERT INTO T1 VALUES (0, 1);

INSERT INTO T1 VALUES (1, 2);

INSERT INTO T1 VALUES (2, 3);

INSERT INTO T1 VALUES (3, 2);

INSERT INTO T1 VALUES (4, 1);

INSERT INTO T1 VALUES (5, 2);

INSERT INTO T1 VALUES (6, 3);

INSERT INTO T1 VALUES (7, 1);


The following query produces the following output:
SELECT C1,C2 FROM T1 ORDER BY C1 SERIES BY C2>1;
C1          C2
---------------------------
1           2
2           3
3           2
5           2
6           3

If you want to know the RANGE value of C1 where the value of the C2 column is larger than 1, you can
determine the range by outputting to which group each record is included with the SERIESNUM function.
```

LIMIT

The LIMIT clause is used to limit the number of records to be output. You can specify an integer to output from the first row to the last row of the result set

```
LIMIT [offset,] row_count

select id1, avg(id2) from exptab where id2 group by id1 order by id1 LIMIT 10;
```

DURATION

DURATION is a keyword that allows you to easily determine the data retrieval scope based on _arrival_time. Used with the BEFORE statement to set a specific range of data at a specific point in time. By using this DURATION, search performance can be dramatically increased and the system load can be dramatically reduced. For more detailed usage, please refer to the following.

```
DURATION Number TimeSpec [BEFORE/AFTER Number TimeSpec]
TimeSpec : YEAR | MONTH | WEEK |  DAY | HOUR | MINUTE | SECOND
```

```
create table t8(i1 integer);
insert into t8 values(1);
insert into t8 values(2);

select i1 from t8;

# Without BEFORE clause
select i1 from t8 duration 2 second;
select i1 from t8 duration 1 minute;
select i1 from t8 duration 1 hour;
select i1 from t8 duration 1 day;
select i1 from t8 duration 1 week;
select i1 from t8 duration 1 month;
select i1 from t8 duration 1 year;

# Using full DURATION statement
select i1 from t8 duration 1 second before 1 day;
select i1 from t8 duration 1 minute before 1 day;
select i1 from t8 duration 1 hour before 1 day;
select i1 from t8 duration 1 day before 1 day;
select i1 from t8 duration 1 week before 1 day;
select i1 from t8 duration 1 month before 1 day;
select i1 from t8 duration 1 year before 1 day;
```

The results are as follows.

```
Mach> create table t8(i1 integer);
Created successfully.

Mach> insert into t8 values(1);
1 row(s) inserted.

Mach> insert into t8 values(2);
1 row(s) inserted.

Mach> select i1 from t8;
i1
--------------
2
1
[2] row(s) selected.

# Without BEFORE clause
Mach> select i1 from t8 duration 2 second;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 minute;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 hour;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 day;
i1
--------------
2
```

```
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 week;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 month;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 year;
i1
--------------
2
1
[2] row(s) selected.

# Using full DURATION statement
Mach> select i1 from t8 duration 1 second before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 minute before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 hour before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 day before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 week before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 month before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 year before 1 day;
i1
--------------
[0] row(s) selected.
```

SAVE DATA

Saves the results of the query directly into the CSV data file.

```
SAVE DATA INTO 'file_name.csv' [{FIELDS | COLUMNS} [TERMINATED BY 'char'] [ENCLOSED BY 'char'] ] [HEADER
ON|OFF] [ENCODED BY coding_name] AS select query;
```

The options are described below.

Options	Description
(FIELDS|COLUMNS) TERMINATED BY 'term_char' ENCLOSED BY 'escape_char'	Specifies the column delimiter and escape delimiter of the csv file to be created.
HEADER (ON|OFF)	Decides whether to enter the column name on the first line of the csv file to be created. The default is OFF.
ENCODED BY coding_name coding_name = (UTF8, MS949, KSC5601, EUCJP, SHIFTJIS, BIG5, GB231280)	Specifies the encoding format of the output data file. The default value is UTF8.

```
SAVE DATA INTO '/tmp/aaa.csv' AS select * from t1;
-- Execute select statement and write result to '/tmp/aaa.csv' file in csv format.

SAVE DATA INTO '/tmp/ccc.csv'  FIELDS TERMINATED BY ';' ENCLOSED '\''  HEADER ON ENCODED BY 'MS949' AS
select * from t1 where i1 > 100;
-- Execute select statement and write result to /tmp/ccc.csv file. Specify field separator and escape
separator, respectively, and set encoding of stored data to MS949.
```