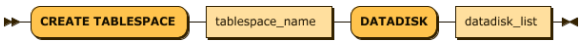
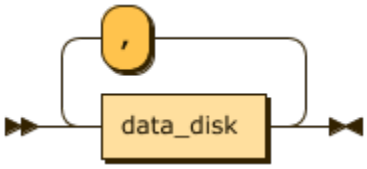
DDL

CREATE TABLESPACE

create_tablespace_stmt:

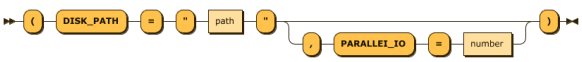


datadisk_list:



data_disk:

data_disk_property:




```
create_tablespace_stmt ::= 'CREATE TABLESPACE' tablespace_name 'DATADISK'
datadisk_list
datadisk_list ::= data_disk ( ',' data_disk )*
data_disk ::= disk_name data_disk_property
data_disk_property ::= '(' 'DISK_PATH' '=' '"' path '"' ( ','
'PARALLEL_IO' '=' number )? ')'
```

```
-- Example
create tablespace tbs1 datadisk disk1 (disk_path=""); -- $MACHBASE_HOME
/dbs/  (Create in $MACHBASE_HOME/dbs/)
create tablespace tbs1 datadisk disk1 (disk_path="tbs1_disk1"); --
$MACHBASE_HOME/dbs/tbs1_disk1  (Created in $MACHBASE_HOME/dbs/tbs1_disk1.
tbs1_disk1 folder must exist)
create tablespace tbs2 datadisk disk1 (disk_path="tbs2_disk1", parallel_io
= 5);
create tablespace tbs1 datadisk disk1 (disk_path="tbs1_disk1", parallel_io
= 10), disk2 (disk_path="tbs1_disk2"), disk3 (disk_path="tbs1_disk3");
```

The CREATE TABLESPACE statement creates a tablespace in $MACHBASE_HOME/dbs/ where the indexes of the log table or log table will be stored.

Tablespace can have multiple disks. When each Partition File that stores data of Table and Index is stored, it is distributed and stored in Data Disks belonging to Tablespace.
If two or more disks are used, the index and table files are distributed and stored on each disk, and I/O is performed in parallel on each device. As the number of disks increases, disk I / O throughput increases, and a large amount of data can be stored on the disk quickly
Also, if tables and index tablespace are separately created and different disks are defined, I/O of table and index can be logically separated without reconfiguration of physical disk.

DATA DISK

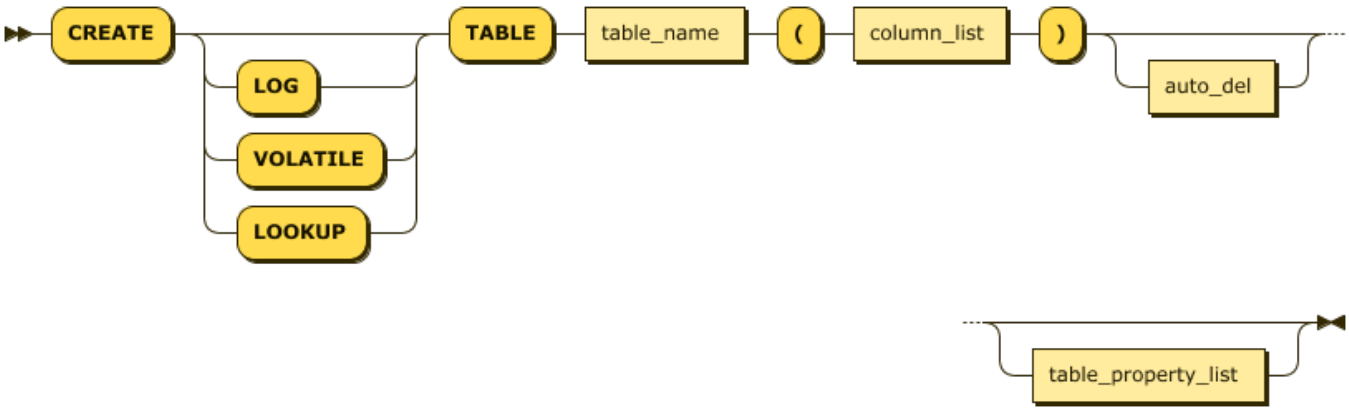
Defines disk belonging to a tablespace. Each Disk has the following properties.

Property	Description

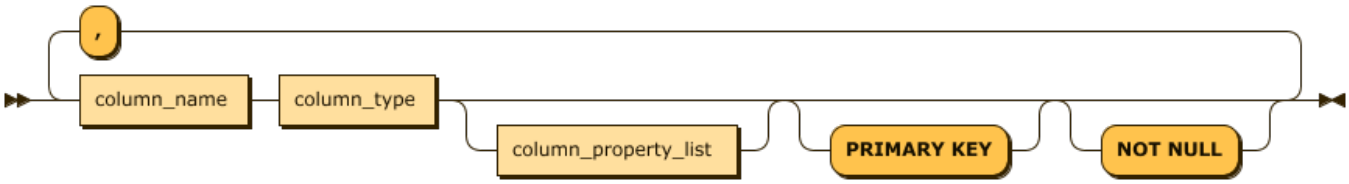
disk_name	Specifies the name of the Disk object. It is used to change the attributes of the Disk object through Alter Tablespace syntax later.
data_disk_property	Specifies the attributes of the disk.
disk_path	Specifies the Directory Path of the disk. This Directory must be created. When a path is specified as a relative path, PATH is searched based on $MACHBASE_HOME/dbs. For example, if PATH = 'disk1', Disk Path is recognized as $MACHBASE_HOME/dbs /disk1.
parallel_io	Determines how many disk IO requests are allowed to be paralleled. (DEF: 3, MIN: 1, MAX: 128)

CREATE TABLE

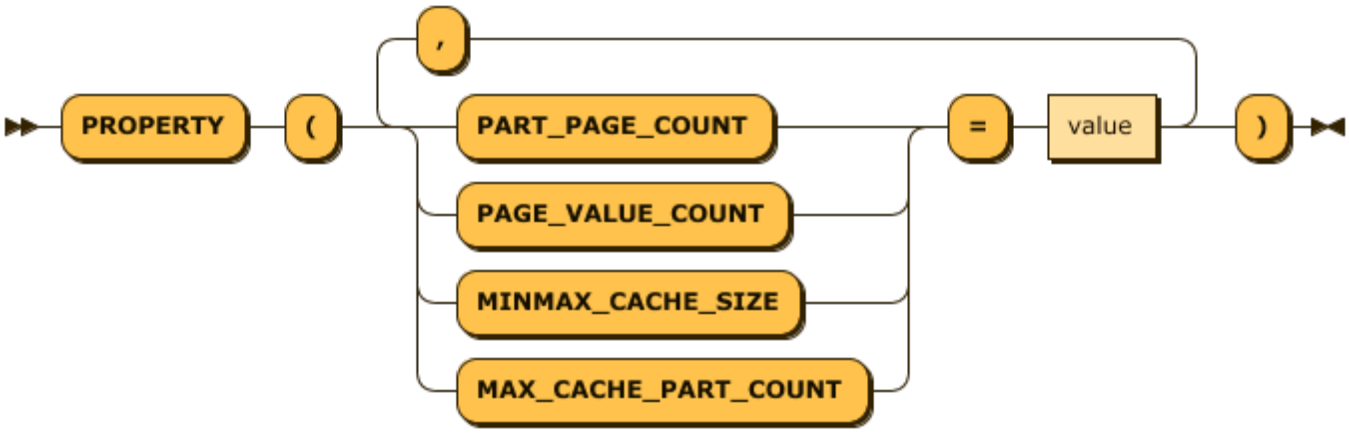
create_table_stmt:



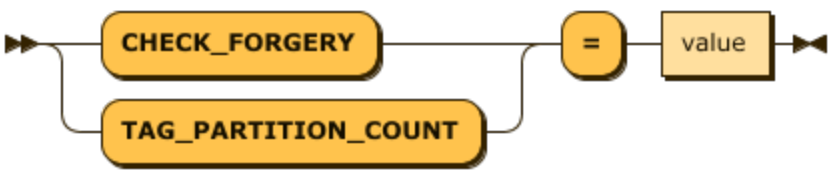
column_list:



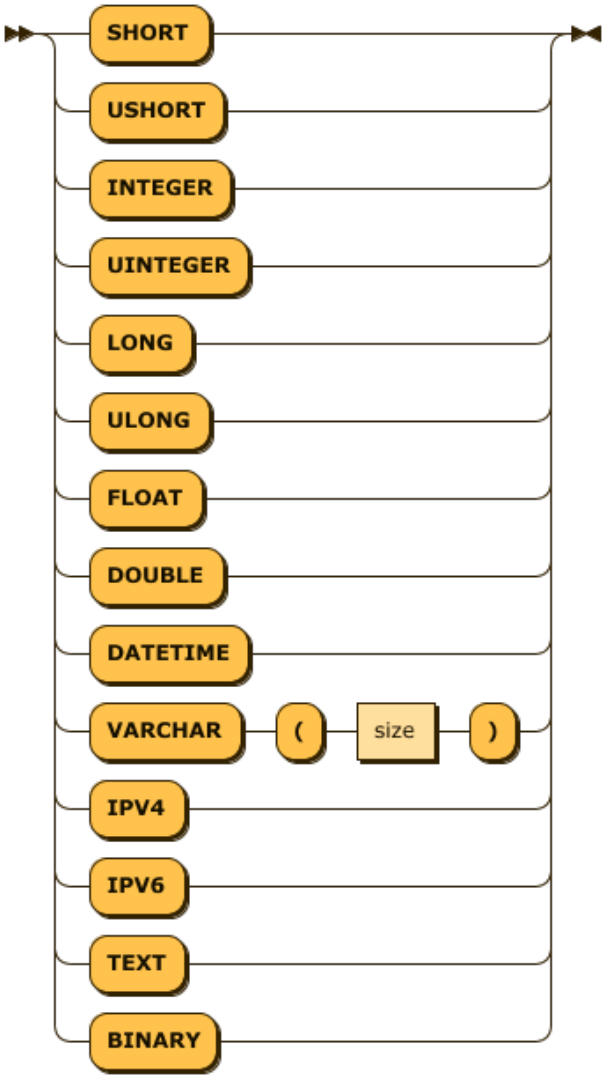
column_property_list:



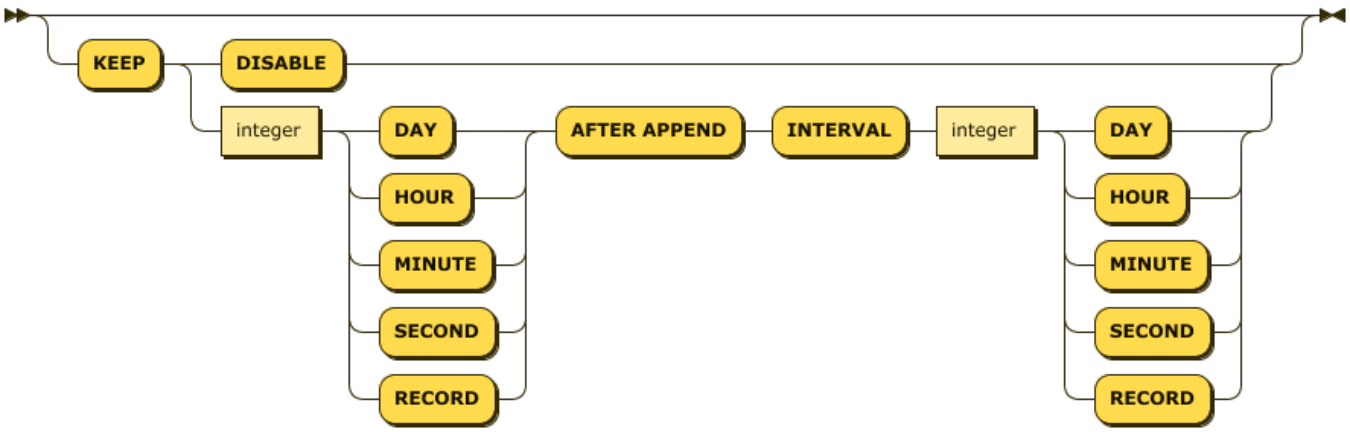
table_property_list:



column_type:



auto_del:



```
create_table_stmt ::= 'CREATE' ( 'LOG' | 'VOLATILE' | 'LOOKUP' )? 'TABLE' table_name '(' column_list ')'
table_property_list?
column_list ::= column_name column_type column_property_list? 'PRIMARY KEY'? 'NOT NULL'? ( ',' column_name
column_type column_property_list? 'PRIMARY KEY'? 'NOT NULL'? )*
column_property_list ::= 'PROPERTY' '(' ( 'PART_PAGE_COUNT' | 'PAGE_VALUE_COUNT' | 'MINMAX_CACHE_SIZE' |
'MAX_CACHE_PART_COUNT' ) '=' value ( ',' ( 'PART_PAGE_COUNT' | 'PAGE_VALUE_COUNT' | 'MINMAX_CACHE_SIZE' |
'MAX_CACHE_PART_COUNT' ) '=' value )* ')'
table_property_list ::=  ( 'CHECK_FORGERY' | 'TAG_PARTITION_COUNT' ) '=' value
column_type ::= 'SHORT' | 'USHORT' | 'INTEGER' | 'UINTEGER' | 'LONG' | 'ULONG' | 'FLOAT' | 'DOUBLE' |
'DATETIME' | 'VARCHAR' '(' size ')' | 'IPV4' | 'IPV6' | 'TEXT' | 'BINARY'
```

```
-- Create ctest table with 5 columns
Mach> create table ctest (id integer, name varchar(20), sipv4 ipv4,dipv6 ipv6,comment text);
Created successfully.
```

Table Type

Table Type	Description
LOG_TABLE	If there is no keyword between CREATE TABLE, a log table is created.
VOLATILE_TABLE	VOLATILE_TABLE is a temporary table in which all data resides in temporary memory and joins the log table to improve the results, The Machbase server disappears as soon as it is shut down.
LOOKUP_TABLE	Like VOLATILE_TABLE, LOOKUP_TABLE can perform fast query processing by storing all the data in memory.

Data Column

A table column is usually a string, but it can be enclosed in single quotes (') or double quotes (") to contain special characters.

```
Mach> CREATE TABLE special_tbl ( with.dot INTEGER );
[ERR-02010: Syntax error: near token (.dot INTEGER )).]
CREATE TABLE special_tbl ( "with.dot" INTEGER ); -- (Possible)
CREATE TABLE special_tbl ( 'with.dot' INTEGER ); -- (Possible)
```

When querying the column through a SELECT query, it must be enclosed in either single or double quotes.

- When enclosed in single quotation marks, the name of the SELECT query result column is output with single quotation marks.
- When enclosed in double quotes, the name of the SELECT query result column is output as it is.

```
Mach> SELECT 'with.dot' FROM special_tbl;
'with.dot'
--------------
[0] row(s) selected.

Mach> SELECT "with.dot" FROM special_tbl;
with.dot
--------------
[0] row(s) selected.
```

Table Property

Specifies the attributes for the table.

Property Name	Available Table Types
CHECK_FORGERY	LOG TABLE

CHECK_FORGERY(Default:0)

This is a property supported only for Disk Column Table. If appended data is maliciously changed later, it can check whether data is changed or not. C heck the ALTER TABLE CHECK FORGERY RESULTFILE syntax.

Column Property

Specifies the attribute for the column.

Property Name	Available Table Types
PART_PAGE_COUNT	LOG TABLE
PAGE_VALUE_COUNT	LOG TABLE
MAX_CACHE_PART_COUNT	LOG TABLE
MINMAX_CACHE_SIZE	LOG TABLE

PART_PAGE_COUNT

This property represents the number of pages a partition has. The number of values that a partition has is PART_PAGE_COUNT * PAGE_VALUE_COUNT.

PAGE_VALUE_COUNT

This property represents the number of values that a page has.

MAX_CACHE_PART_COUNT (Default : 0)

This property sets the cache area for performance.

When Machbase accesses a partition, it first looks for a structure that contains the meta information of that partition in memory. It determines how many partition information it contains in memory. Larger size will help performance, but memory usage will increase. The minimum value is 1 and the maximum value is 65535.

MINMAX_CACHE_SIZE (Default : 10240)

This property specifies how much cache memory to use for the MINMAX of the corresponding column. The default is 100MB for _ARRIVAL_TIME, the 0th hidden column. However, other columns are specified as 10KB by default. This size can be changed after the creation of the table through the "ALTER TABLE MODIFY" statement.

NOT NULL Constraint

Specifies NOT NULL if the column value does not allow NULL, and omit it if it is allowed (Default).

You can change the constraint with the ALTER TABLE MODIFY COLUMN command to drop or add this constraint defined after the creation of the table.

```
# Column c1 is not null and c2 is created without not null constraint.
CREATE TABLE t1(c1 INTEGER NOT NULL, c2 VARCHAR(200));
```

Pre-defined System Columns

When you create a table using the Create Table statement, the system creates two additional predefined system columns. _ARRIVAL_TIME and _RID columns.

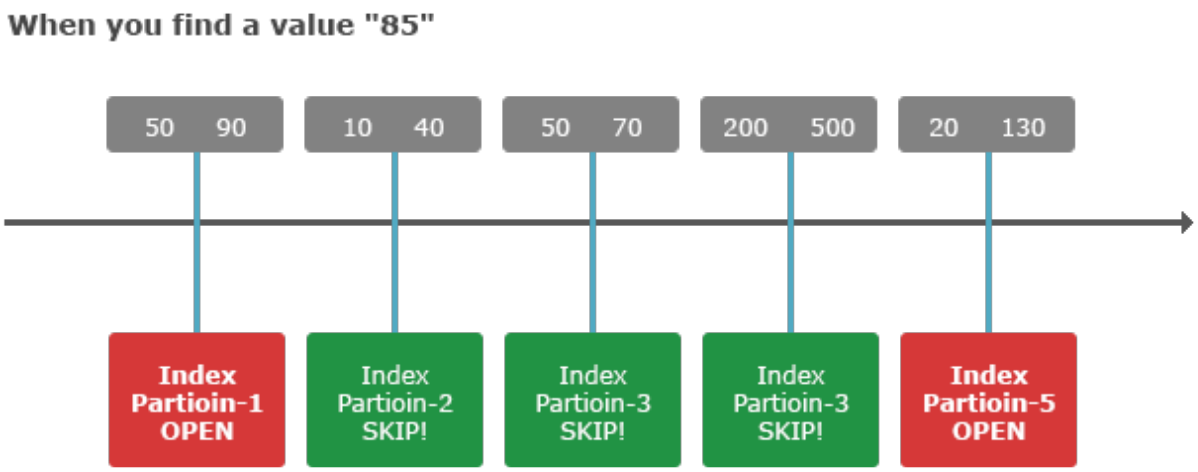
The _ARRIVAL_TIME column is inserted into the DATETIME column based on the system time at which data is inserted into the INSERT statement or AppendData, and the value can be used as the unique key of the generated record. The value of this column can be inserted by specifying the value in the machloader or INSERT statement if the order is guaranteed (in the order of past-present). When data is retrieved using the DURATION conditional expression, data is retrieved based on the value of this column.
The _RID column is created by the system as a unique value for a particular record. The data type of this column is a 64-bit integer. For this column, the user can not specify a value and can not create an index. It is automatically generated at the time of data INSERT. You can retrieve records by the value of the _RID column.

```
create volatile table t1111 (i1 integer);
Created successfully.
Mach> desc t1111;

----------------------------------------------------------------
NAME                          TYPE                LENGTH
----------------------------------------------------------------
_ARRIVAL_TIME                 datetime            8
I1                            integer             4

Mach>insert into t1111 values (1);
1 row(s) inserted.
Mach>select _rid from t1111;
_rid
----------------------
0
[1] row(s) inserted.

Mach>select i1 from t1111 where _rid = 0;
i1
--------------
1
[1] row(s) selected.
```

MINMAX Cache Concept

In general, in the Disk DBMS, when a specific value is searched using the index, the disk is accessed to access the disk area including the index, and the final disk page including the corresponding value is searched.

On the other hand, Machbase is a chronologically partitioned structure in order to maintain time series information, which means that a particular piece of index information is divided into chunks of files in chronological order. Therefore, when a Machbase index is used, an index file fragmented by such a partition is sequentially searched.
If the range of data to be searched is divided into 1000 partitions, it means that 1000 files should be opened and retrieved every time. Although it is designed as an efficient columnar database structure, the MINMAX_CACHE structure is a way to improve the performance because the I/O cost is proportional to the number of index partitions.
MINMAX_CACHE is a structure that holds the index file information of the partition in memory, and is a contiguous memory space that keeps the minimum and maximum values of the column in memory. By maintaining such structure, when a partition containing a specific value is searched, if the value is smaller than the minimum value of the index or is larger than the maximum value, the corresponding partition can be skipped altogether, thereby enabling high-performance data analysis.



As shown in the figure above, to find the value 85, only the partitions 1 and 5 included in MIN/MAX among the 5 partitions are actually searched, and the partitions 2, 3 and 4 are skipped altogether.

MINMAX Cache Column

You can decide whether to use MINMAX Cache for a particular column when creating the table.

If the minmax_cache_size is set to a value other than 0, the MINMAX Cache will be active when the index is searched for that column and will not be active if `MINMAX_CACHE_SIZE` = 0.
Please note the following when using this MINMAX Cache.

1. MINMAX Cache does not need to explicitly create an index on the column.
2. As default for all columns, `MINMAX_CACHE_SIZE` is set to 10KB and the Alter Table syntax can be used to reset the memory size to a reasonable size.
3. The hidden column _arrival_time is 100MB by default and automatically uses MINMAX Cache memory.
4. In the case of VARCHAR type, MINMAX Cache is not covered. Therefore, if you explicitly specify whether the VARCHAR type is cached, an error will occur.
5. When the corresponding table is created, the `MINMAX_CACHE_SIZE` maximum memory can be used as much as the property is set. As the number of partitions grows, the memory grows gradually and increases by the maximum memory above.
6. If there are no records in the table, MINMAX Cache memory is not allocated at all.

Below is an example of table creation using actual MINMAX.

```
-- MINMAX_CACHE_SIZE = 0 for VARCHAR is allowed semantically.
CREATE TABLE ctest (id INTEGER, name VARCHAR(100) PROPERTY(MINMAX_CACHE_SIZE = 0));
Created successfully.
Mach>

-- Cache applied to id column.
CREATE TABLE ctest2 (id INTEGER PROPERTY(MINMAX_CACHE_SIZE = 10240), name VARCHAR(100) PROPERTY
(MINMAX_CACHE_SIZE = 0));
Created successfully.
Mach>

-- Applied to id1, id2, and id3.
CREATE TABLE ctest3 (id1 INTEGER PROPERTY(MINMAX_CACHE_SIZE = 10240), name VARCHAR(100) PROPERTY
(MINMAX_CACHE_SIZE = 0), id2 LONG PROPERTY(MINMAX_CACHE_SIZE = 1024), id3 IPV4 PROPERTY(MINMAX_CACHE_SIZE =
1024), id4 SHORT);
Created successfully.
Mach>

-- MINMAX_CACHE_SIZE is specified in column units or set to 0.
CREATE TABLE ctest4 (id1 INTEGER PROPERTY(MINMAX_CACHE_SIZE=10240), name VARCHAR(100) PROPERTY
(MINMAX_CACHE_SIZE=0), id2 LONG PROPERTY(MINMAX_CACHE_SIZE=10240), id3 IPV4 PROPERTY(MINMAX_CACHE_SIZE=0),
id4 SHORT);
Created successfully.
Mach>
```

Primary Key

This is a constraint that can be assigned to a Volatile/Lookup table column. The Volatile / Lookup table does not always need to have a primary key, but you can not use the INSERT ON DUPLICATE KEY UPDATE statement without a primary key.

When a primary key is assigned, a red-black tree index corresponding to the primary key is generated.

AUTO_DEL Clauses

Supported version 5.5 or later.

This feature limits the amount of data to maintain disk storage space. It is only supported for log table. It is set by using AUTO_DEL clause before specifying table property in CREATE TABLE statement. The AUTO_DEL clause can be set based on the storage time or the number of records.

```
CREATE TABLE t1 (c1 INT) KEEP 30 DAY AFTER APPEND INTERVAL 5 SECOND;
```

The above example deletes data that is more than 30 days old if there is more input five seconds after the automatic deletion is performed. If the interval specified is too long, auto delete is performed for a long time, which may affect the input. If it is too short, it may affect overall system performance.

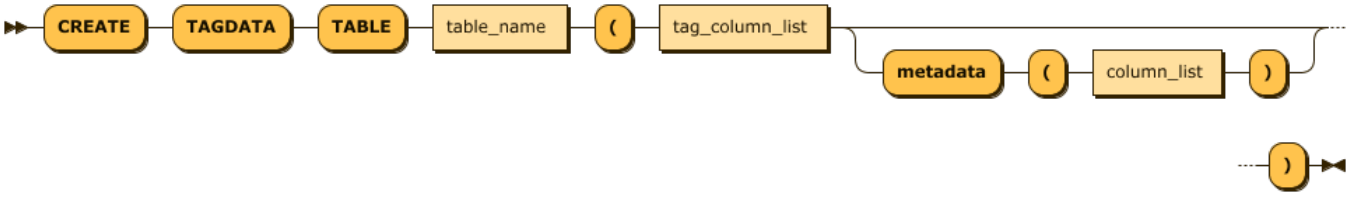
The following example shows how to specify the automatic deletion function as the number of data to save.

```
CREATE TABLE t1 (c1 INT) KEEP 3 RECORD AFTER APPEND INTERVAL 5 RECORD;
```

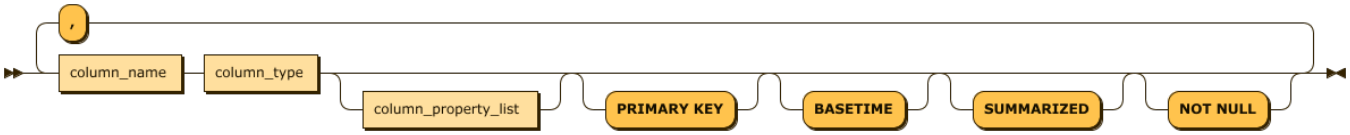
It checks the number of records in the table for every 5 input data and automatically deletes it and keep only 3 records if there are more than three.

CREATE TAG TABLE

create_tag_table_stmt:



tag_column_list:



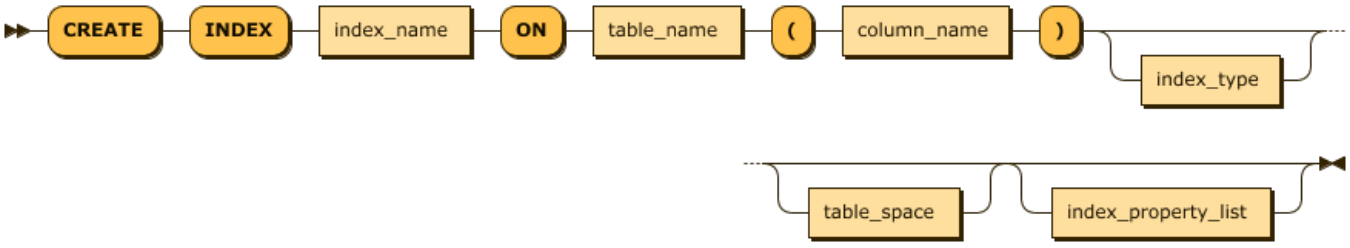

```
create_tag_table_stmt ::= 'CREATE' 'TAGDATA' 'TABLE' table_name '(' tag_column_list ( 'metadata' '('
column_list ')' )? ')'
tag_column_list ::= column_name column_type column_property_list? 'PRIMARY KEY'? 'BASETIME'? 'SUMMARIZED'?
'NOT NULL'? ( ',' column_name column_typecolumn_property_list? 'PRIMARY KEY'? 'BASETIME'? 'SUMMARIZED'? 'NOT
NULL'? )*
```

The primary key, basetime, and summarized must be included in the tag table creation.

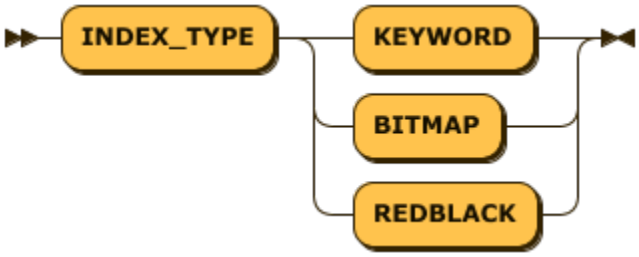
```
-- Example
CREATE TAGDATA TABLE tag (name varchar(20) primary key, time datetime basetime, value double summarized);
CREATE TAGDATA TABLE tag (name varchar(20) primary key, int_column int, time datetime basetime, value double
summarized, value2 float);
CREATE TAGDATA TABLE tag (name varchar(20) primary key, time datetime basetime, value double summarized,
value2 float) METADATA (i1 int);
```

CREATE INDEX

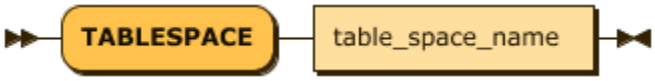
create_index_stmt:



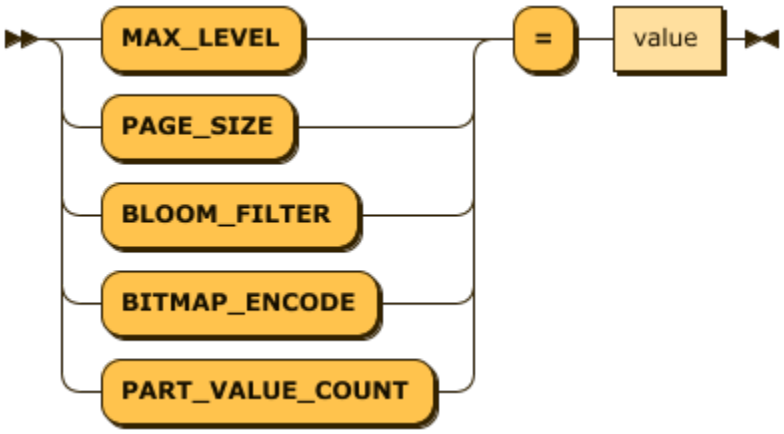
index_type:



table_space:



index_property_list:




```
create_index_stmt ::= 'CREATE' 'INDEX' index_name 'ON' table_name '(' column_name ')' index_type?
table_space? index_property_list?
index_type ::= 'INDEX_TYPE' ( 'KEYWORD' | 'BITMAP' | 'REDBLACK' )
table_space ::= 'TABLESPACE' table_space_name
index_property_list ::= ( 'MAX_LEVEL' | 'PAGE_SIZE' | 'BLOOM_FILTER' | 'BITMAP_ENCODE' | 'PART_VALUE_COUNT'
) '=' value
```

Index Type

Specifies the Index Type to be created. If it is not Keyword Index, Index Type is created as Default Index Type according to Table Type if Index Type is not specified.

Table Type	Default Index Type
Volatile Table	REDBLACK
Lookup Table	REDBLACK
Log Table	LSM

KEYWORD Index

This can be created only for varchar and text column of log table. It can be created for only one column.

LSM Index

LSM (Log Structure Merge) Index is an index optimized for storing and searching Big Data. The partitions of the LSM indexes are maintained for each level, and the lower level partitions are merged to move to the upper level. Lower partitions used to create a higher level partition are deleted.

This Index Level Partition Building is performed by Background Thread. The upper level partitions are merged with the lower level partitions and are created as one partition, so there are the following advantages when searching through the index.

1. If the key is duplicated, the disk space for key storage is saved because it is stored only once.
2. Searching for multiple partitions reduces the cost of opening and closing the file when searching for one index partition, and the number of index pages accessed is also reduced.

LSM Index Property

Item	Description
MAX_LEVEL (DEFAULT = 3, MIN = 0, MAX = 3)	The maximum level of the LSM Index, and the current value of 3 is the maximum value. And the maximum number of records of one partition can not exceed 200 million. The partition size of each level is the number of values of the previous partition * 10. For example, if MAX_LEVEL = 3 and PART_VALUE_COUNT is 100,000, then Level 0 = 100,000, Level 1 = 1,000,0000, Level 2 = 10,000,000, and Level 3 = 100,000,000. If the Partition Size of the last level exceeds 200 million, index creation will fail.
PAGE_SIZE (DEFAULT = 512 * 1024, MIN = 32 * 1024, MAX = 1 * 1024 * 1024)	Specifies the size of the page in which the index key value and bitmap value are stored. Default is 512K.
BLOOM_FILTER (DEFAULT = 1, DISABLE = 0, ENABLE (DEFAULT) = non_zero_integer)	Sets whether to set Bloom filter on index. Setting the Bloom filter allows you to quickly search for values that do not exist in the index, but it will increase the time it takes to create the index. If you use only Range conditions, Bloom filters are not available and need not be created. If you do not want to create a Bloom filter, you should set BLOOM_FILTER = 0.
BITMAP_ENCODE (DEFAULT = EQUAL, RANGE)	Sets the bitmap type of the index. If BITMAP_ENCODE = EQUAL (default), generates a bitmap for the same value as the key value. If BITMAP = RANGE, generates a bitmap according to the range of the key value. It is better to set as BITMAP_ENCODE = EQUAL when using = as the query condition, and BITMAP_ENCODE = RANGE when using the specific range value as the query condition. In the case of BITMAP = RANGE, the cost of creation increases slightly compared to EQUAL.

BITMAP Index

This is an index for data analysis and can be created only in the log table. It can be created on all columns except varchar, text, and binary, and can only be created on a single column.

RED-BLACK Index

This is a memory index for real-time data retrieval. It can be created only in the Volatile/Lookup table. It can be created in all columns of this table and can only be created for a single column.

Index Property

The properties that can be applied in the LSM Index are as follows.

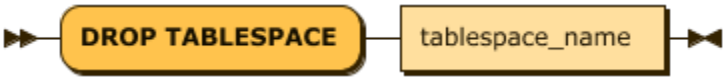
PART_VALUE_COUNT

Indicates the number of rows stored in the Partition of Index.

```
-- Example
-- Index applied to c1 column.
CREATE INDEX index1 on table1 ( c1 )
-- Keyword index applied to var_column of varchar type, and page_size unit is 100000.
CREATE INDEX index2 on table1 (var_column) INDEX_TYPE KEYWORD PAGE_SIZE=100000;
```

DROP TABLESPACE

drop_table_stmt:



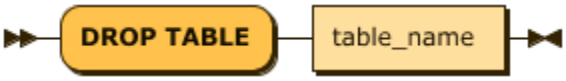

```
drop_table_stmt ::= 'DROP TABLESPACE' tablespace_name
```

Deletes the specified tablespace. However, if the object created in Tablespace exists, deletion fails.

```
-- Example
DROP TABLESPACE TablespaceName;
```

DROP TABLE

drop_table_stmt:



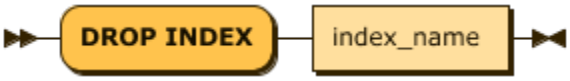

```
drop_table_stmt ::= 'DROP TABLE' table_name
```

Deletes the specified table. However, if there is another session in which the table is being searched, it fails with an error.

```
-- Example
DROP TABLE TableName;
```

DROP INDEX

drop_index_stmt:




```
drop_index_stmt ::= 'DROP INDEX' index_name
```

Deletes the specified index. However, if there is another session in which the table is being searched, it fails with an error.

```
-- Example
DROP INDEX IndexName;
```

ALTER TABLESPACE

The ALTER TABLESPACE statement is used to change the information associated with the specified tablespace.

ALTER TABLESPACE MODIFY DATADISK

This syntax is used to change the properties of DATADISK in Tablespace.

alter_tablespace_stmt:


```
alter_tablespace_stmt ::= 'ALTER TABLESPACE' table_name 'MODIFY DATADISK' disk_name 'SET' 'PARALLEL_IO' '='
value
```

```
-- Example
ALTER TABLESPACE tbs1 MODIFY DATADISK disk1 SET PARALLEL_IO = 10;
```

ALTER TABLE

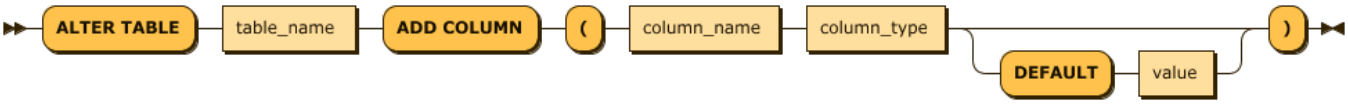
The ALTER TABLE statement is used to change the schema information of the specified table, and only the Log Table is available.

ALTER TABLE SET

This syntax changes the properties of a table. Currently there are no dynamically changeable properties.

ALTER TABLE ADD COLUMN

alter_table_add_stmt:




```
alter_table_add_stmt ::= 'ALTER TABLE' table_name 'ADD COLUMN' '(' column_name column_type ( 'DEFAULT' value
)? ')'
```

This syntax is the ability to add a specific column to the table in real time. You can add the name and type of the column, and set the default data values through the DEFAULT clause.

```
-- Example-1
alter table atest2 add column (id4 float);

-- Example-2
alter table atest2 add column (id6 double  default 5);
alter table atest2 add column (id7 ipv4  default '192.168.0.1');
alter table atest2 add column (id8 varchar(4) default 'hello');
```

ALTER TABLE DROP COLUMN

alter_table_drop_stmt:


```
alter_table_drop_stmt ::= 'ALTER TABLE' table_name 'DROP COLUMN' '(' column_name ')'
```

This syntax is to delete a specific column in the table in real time.

```
-- Example
alter table atest2 drop column (id4);
alter table atest2 drop column (id8);
```

ALTER TABLE RENAME COLUMN

alter_table_column_rename_stmt:

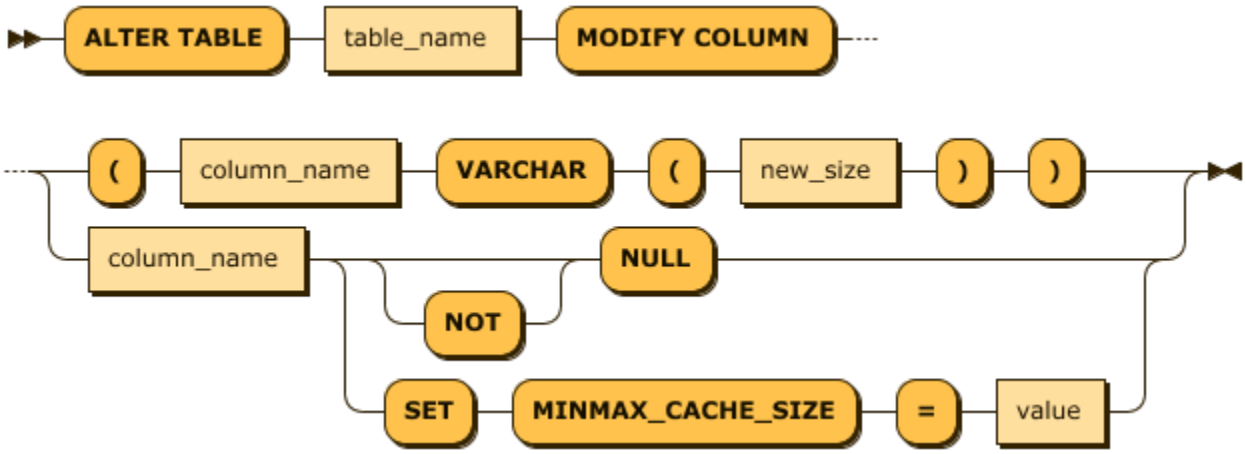

```
alter_table_column_rename_stmt ::= 'ALTER TABLE' table_name 'RENAME
COLUMN' old_column_name 'TO' new_column_name
```

This syntax is a function that changes a specific column name in a table.

```
-- Example
alter table atest2 rename column id7 to id7_rename;
```

ALTER TABLE MODIFY COLUMN

alter_table_modify_stmt:




```
alter_table_modify_stmt ::= 'ALTER TABLE' table_name 'MODIFY COLUMN' ( '(' column_name 'VARCHAR' '('
new_size ')' ')' | column_name ( 'NOT'? 'NULL' | 'SET' 'MINMAX_CACHE_SIZE' '=' value ) )
```

This syntax changes the properties of a particular column of a table. Currently it is possible to modify MINMAX CACHE attributes and NOT NULL constraints for column lengths and other types of VARCHAR types.

VARCHAR SIZE

This syntax supports changing the column length of VARCHAR type only. This operation can not be reduced in length to preserve existing data, and should always be increased.

```
ALTER TABLE table_name MODIFY COLUMN (column_name VARCHAR(new_size));
```

```
-- Example: Assume TABLE is created like this.
-- create table atest5 (id integer, name varchar(5), id3 double, id4 float);

-- Error occurred: Can not change to another type,
alter table atest5 modify column (id varchar(10));

-- Error occurred: VARCHAR length can not be made smaller.
alter table atest5 modify column (name varchar(3));

-- Error occurred: Maximum size of VARCHAR can not exceed 32767.
alter table atest5 modify column (name varchar(32768));

-- Success
alter table atest5 modify column (name varchar(128));
```

MINMAX_CACHE_SIZE

This syntax changes MINMAX_CACHE_SIZE for a particular column.

```
ALTER TABLE table_name MODIFY COLUMN column_name SET MINMAX_CACHE_SIZE=value;
```

```
-- Example: Assume TABLE is created like this.
create table atest9 (id integer, name varchar(100));

-- Error: Does not apply to VARCHAR.
alter table atest9 modify column name set minmax_cache_size=0;
[ERR-02139 : MINMAX CACHE is not allowed for VARCHAR column(NAME).]

-- Change success
alter table atest9 modify column id set minmax_cache_size=10240;
```

NOT NULL

Adds a NOT NULL constraint to the column. If you add a NOT NULL constraint, the DDL operation fails for columns with NULL values.

If you want to allow NULL values in a column, use the MODIFY COLUMN NULL command in the next section.

```
ALTER TABLE table_name MODIFY COLUMN column_name NOT NULL;
```

```
-- Add NOT NULL constraint to t1.c1.
alter table t1 modify column c1 not null;
```

NULL

Releases the NOT NULL constraint. Performance improvement due to min_max cache of LSM index can not be obtained. NULL values can be input.

```
ALTER TABLE table_name MODIFY COLUMN column_name NULL;
```

```
-- Release NOT NULL constraint at t1.c1.
alter table t1 modify column c1 null;
```

ALTER TABLE FLUSH

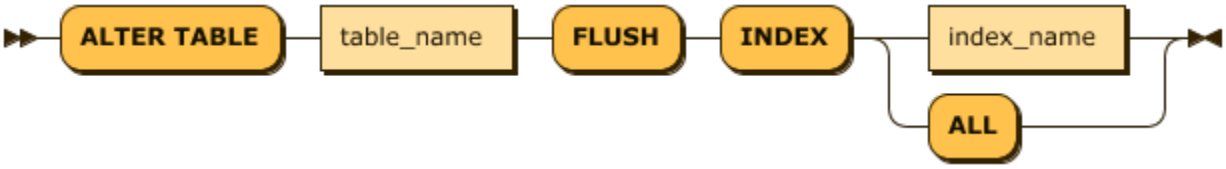
alter_table_flush_stmt:


```
alter_table_flush_stmt ::= 'ALTER TABLE' table_name 'FLUSH'
```

Waits until the input data for the specified table is fully reflected in the data file.

ALTER TABLE FLUSH INDEX

alter_table_flush_index_stmt:




```
alter_table_flush_index_stmt ::= 'ALTER TABLE' table_name 'FLUSH' 'INDEX' ( index_name | 'ALL' )
```

Waits until the index data of the specified table is completely reflected in the index file.

```
-- Wait until data for all indexes of bulktable is reflected in index files.
alter table bulktable flush index all;

-- Wait until data of c1_idx index of bulktable is reflected in index file.
alter table bulktable flush index c1_idx;
```

ALTER TABLE CHECK FORGERY RESULT FILE

alter_table_check_forgery_stmt:

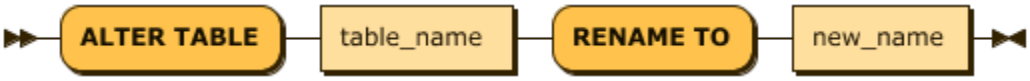

```
alter_table_check_forgery_stmt ::= 'ALTER TABLE' table_name 'CHECK FORGERY RESULTFILE' "'" result_filename
"'"
```

Checks whether the table has changed since it was appended to the specified table (Forgery). Forgery Check is only supported for Disk Column Table, and when creating Disk Column Table, it should be created by setting FORGERY_CHECK attribute to 1. When the Forgery is detected, a file corresponding to result_filename is created under $MACHBASE_HOME/trc and the file shows which part of the table has been changed.

```
-- Perform forgery check on t1 table.
alter table t1 check forgery resultfile 't1_forgery_check';
```

ALTER TABLE RENAME TO

alter_table_rename_stmt:




```
alter_table_rename_stmt ::= 'ALTER TABLE' table_name 'RENAME TO' new_name
```

Changes the name of the table.

Metatables can not be renamed, and you can not use the $ character in the name to be changed. Table renaming is only possible for log tables.

```
-- Change name of user table to employee.
ALTER TABLE user RENAME TO employee
```

TRUNCATE TABLE

truncate_table_stmt:


```
truncate_table_stmt ::= 'TRUNCATE TABLE' table_name
```

```
-- Delete all data in ctest table.
Mach> truncate table ctest;
Truncated successfully.
```

Deletes all data in the specified table. However, if there is another session in which the table is being searched, it fails with an error.