尚硅谷大数据技术之ClickHouse

(作者：尚硅谷大数据研发部)

版本：V1.0

第1章 ClickHouse概述

1.1 什么是ClickHouse

ClickHouse 是俄罗斯的Yandex于2016年开源的列式存储数据库（DBMS），主要用于在线分析处理查询（OLAP），能够使用SQL查询实时生成分析数据报告。

1.2 什么是列式存储

以下面的表为例：

|  |  |  |
| --- | --- | --- |
| Id | Name | Age |
| 1 | 张三 | 18 |
| 2 | 李四 | 22 |
| 3 | 王五 | 34 |

采用行式存储时，数据在磁盘上的组织结构为：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 张三 | 18 | 2 | 李四 | 22 | 3 | 王五 | 34 |

好处是想查某个人所有的属性时，可以通过一次磁盘查找加顺序读取就可以。但是当想查所有人的年龄时，需要不停的查找，或者全表扫描才行，遍历的很多数据都是不需要的。

而采用列式存储时，数据在磁盘上的组织结构为：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 张三 | 李四 | 王五 | 18 | 22 | 34 |

这时想查所有人的年龄只需把年龄那一列拿出来就可以了

1.3 安装前的准备

1.3.1 CentOS取消打开文件数限制

在/etc/security/limits.conf、/etc/security/limits.d/90-nproc.conf这2个文件的末尾加入一下内容：

[root@hadoop102 software]# vim /etc/security/limits.conf

在文件末尾添加：

\* soft nofile 65536

\* hard nofile 65536

\* soft nproc 131072

\* hard nproc 131072

[root@hadoop102 software]# vim /etc/security/limits.d/90-nproc.conf

在文件末尾添加：

\* soft nofile 65536

\* hard nofile 65536

\* soft nproc 131072

\* hard nproc 131072

重启服务器之后生效，用ulimit -n 或者ulimit -a查看设置结果

[root@hadoop102 ~]# ulimit -n

65536

1.3.2 CentOS取消SELINUX

修改/etc/selinux/config中的SELINUX=disabled后重启

[root@hadoop102 ~]# vim /etc/selinux/config

SELINUX=disabled

1.3.3 关闭防火墙

[root@hadoop102 ~]# service iptables stop

[root@hadoop102 ~]# service ip6tables stop

ip6tables：将 chains 设置为 ACCEPT 策略：filter [确定]

ip6tables：清除防火墙规则： [确定]

：正在卸载模块： [确定]

1.3.4 安装依赖

[root@hadoop102 ~]# yum install -y libtool

[root@hadoop102 ~]# yum install -y \*unixODBC\*

第2章 安装

2.1 网址

官网：https://clickhouse.yandex/

下载地址：http://repo.red-soft.biz/repos/clickhouse/stable/el6/

2.2 单机模式

2.2.1 上传5个文件到/opt/software/

[root@hadoop102 software]# ls

clickhouse-client-1.1.54236-4.el6.x86\_64.rpm

clickhouse-server-1.1.54236-4.el6.x86\_64.rpm

clickhouse-compressor-1.1.54236-4.el6.x86\_64.rpm

clickhouse-server-common-1.1.54236-4.el6.x86\_64.rpm

clickhouse-debuginfo-1.1.54236-4.el6.x86\_64.rpm

2.2.2分别安装这5个rpm文件

[root@hadoop102 software]#

rpm -ivh clickhouse-server-common-1.1.54236-4.el6.x86\_64.rpm

Preparing… ########################################### [100%]

1:clickhouse-server-commo########################################### [100%]

[root@hadoop102 software]#

rpm -ivh clickhouse-server-1.1.54236-4.el6.x86\_64.rpm

Preparing… ########################################### [100%]

1:clickhouse-server ########################################### [100%]

[root@hadoop102 software]#

rpm -ivh clickhouse-debuginfo-1.1.54236-4.el6.x86\_64.rpm

Preparing… ########################################### [100%]

1:clickhouse-debuginfo ########################################### [100%]

[root@hadoop102 software]#

rpm -ivh clickhouse-client-1.1.54236-4.el6.x86\_64.rpm

Preparing… ########################################### [100%]

1:clickhouse-client ########################################### [100%]

[root@hadoop102 software]#

rpm -ivh clickhouse-compressor-1.1.54236-4.el6.x86\_64.rpm

Preparing… ########################################### [100%]

1:clickhouse-compressor ########################################### [100%]

2.2.3 启动ClickServer

前台启动：

[root@hadoop102 software]# clickhouse-server --config-file=/etc/clickhouse-server/config.xml

后台启动：

[root@hadoop102 software]# nohup clickhouse-server --config-file=/etc/clickhouse-server/config.xml >null 2>&1 &

[1] 2696

2.2.4 使用client连接server

[root@hadoop102 software]# clickhouse-client

ClickHouse client version 1.1.54236.

Connecting to localhost:9000.

Connected to ClickHouse server version 1.1.54236.

:)

2.3 分布式集群安装

2.3.1 在hadoop103，hadoop104上面执行之前的所有步骤

2.3.2 三台机器修改配置文件config.xml

[root@hadoop102 ~]# vim /etc/clickhouse-server/config.xml

<listen\_host>::</listen\_host>

<!-- <listen\_host>::1</listen\_host> -->

<!-- <listen\_host>127.0.0.1</listen\_host> -->

[root@hadoop103 ~]# vim /etc/clickhouse-server/config.xml

<listen\_host>::</listen\_host>

<!-- <listen\_host>::1</listen\_host> -->

<!-- <listen\_host>127.0.0.1</listen\_host> -->

[root@hadoop104 ~]# vim /etc/clickhouse-server/config.xml

<listen\_host>::</listen\_host>

<!-- <listen\_host>::1</listen\_host> -->

<!-- <listen\_host>127.0.0.1</listen\_host> -->

2.3.3 在三台机器的etc目录下新建metrika.xml文件

[root@hadoop102 ~]# vim /etc/metrika.xml

添加如下内容：

<yandex>

<clickhouse\_remote\_servers>

<perftest\_3shards\_1replicas>

<shard>

<internal\_replication>true</internal\_replication>

<replica>

<host>hadoop102</host>

<port>9000</port>

</replica>

</shard>

<shard>

<replica>

<internal\_replication>true</internal\_replication>

<host>hadoop103</host>

<port>9000</port>

</replica>

</shard>

<shard>

<internal\_replication>true</internal\_replication>

<replica>

<host>hadoop104</host>

<port>9000</port>

</replica>

</shard>

</perftest\_3shards\_1replicas>

</clickhouse\_remote\_servers>

<zookeeper-servers>

<node index="1">

<host>hadoop102</host>

<port>2181</port>

</node>

<node index="2">

<host>hadoop103</host>

<port>2181</port>

</node>

<node index="3">

<host>hadoop104</host>

<port>2181</port>

</node>

</zookeeper-servers>

<macros>

<replica>hadoop104</replica>

</macros>

<networks>

<ip>::/0</ip>

</networks>

<clickhouse\_compression>

<case>

<min\_part\_size>10000000000</min\_part\_size>

<min\_part\_size\_ratio>0.01</min\_part\_size\_ratio>

<method>lz4</method>

</case>

</clickhouse\_compression>

</yandex>

注意：上面标红的地方需要根据机器不同去修改

3.3.4 三台机器启动ClickServer

首先在三台机器开启Zookeeper

前台启动：

[root@hadoop102 software]# clickhouse-server --config-file=/etc/clickhouse-server/config.xml

后台启动：

[root@hadoop102 software]# nohup clickhouse-server --config-file=/etc/clickhouse-server/config.xml >null 2>&1 &

[1] 2696

第3章 数据类型

3.1 整型

固定长度的整型，包括有符号整型或无符号整型。

整型范围（-2n-1~2n-1-1）：

Int8 - [-128 : 127]

Int16 - [-32768 : 32767]

Int32 - [-2147483648 : 2147483647]

Int64 - [-9223372036854775808 : 9223372036854775807]

无符号整型范围（0~2n-1）：

UInt8 - [0 : 255]

UInt16 - [0 : 65535]

UInt32 - [0 : 4294967295]

UInt64 - [0 : 18446744073709551615]

3.2 浮点型

Float32 - float

Float64 – double

建议尽可能以整数形式存储数据。例如，将固定精度的数字转换为整数值，如时间用毫秒为单位表示，因为浮点型进行计算时可能引起四舍五入的误差。

:) select 1-0.9

┌───────minus(1, 0.9)─┐

│ 0.09999999999999998 │

└─────────────────────┘

与标准SQL相比，ClickHouse 支持以下类别的浮点数：

Inf-正无穷：

:) select 1/0

┌─divide(1, 0)─┐

│ inf │

└──────────────┘

-Inf-负无穷：

:) select -1/0

┌─divide(1, 0)─┐

│ -inf │

└──────────────┘

NaN-非数字：

:) select 0/0

┌─divide(0, 0)─┐

│ nan │

└──────────────┘

3.3 布尔型

没有单独的类型来存储布尔值。可以使用 UInt8 类型，取值限制为 0 或 1。

3.4 字符串

**String**

字符串可以任意长度的。它可以包含任意的字节集，包含空字节。

**FixedString(N)**

固定长度 N 的字符串，N 必须是严格的正自然数。当服务端读取长度小于 N 的字符串时候，通过在字符串末尾添加空字节来达到 N 字节长度。 当服务端读取长度大于 N 的字符串时候，将返回错误消息。

与String相比，极少会使用FixedString，因为使用起来不是很方便。

3.5 枚举类型

包括 Enum8 和 Enum16 类型。Enum 保存 'string'= integer 的对应关系。

Enum8 用 'String'= Int8 对描述。

Enum16 用 'String'= Int16 对描述。

用法演示：

创建一个带有一个枚举 Enum8('hello' = 1, 'world' = 2) 类型的列：

CREATE TABLE t\_enum

(

x Enum8('hello' = 1, 'world' = 2)

)

ENGINE = TinyLog

这个 x 列只能存储类型定义中列出的值：'hello'或'world'。如果尝试保存任何其他值，ClickHouse 抛出异常。

:) INSERT INTO t\_enum VALUES ('hello'), ('world'), ('hello')

INSERT INTO t\_enum VALUES

Ok.

3 rows in set. Elapsed: 0.002 sec.

:) insert into t\_enum values('a')

INSERT INTO t\_enum VALUES

Exception on client:

Code: 49. DB::Exception: Unknown element 'a' for type Enum8('hello' = 1, 'world' = 2)

从表中查询数据时，ClickHouse 从 Enum 中输出字符串值。

SELECT \* FROM t\_enum

┌─x─────┐

│ hello │

│ world │

│ hello │

└───────┘

如果需要看到对应行的数值，则必须将 Enum 值转换为整数类型。

SELECT CAST(x, 'Int8') FROM t\_enum

┌─CAST(x, 'Int8')─┐

│ 1 │

│ 2 │

│ 1 │

└─────────────────┘

3.6 数组

**Array(T)：**由 T 类型元素组成的数组。

T 可以是任意类型，包含数组类型。 但不推荐使用多维数组，ClickHouse 对多维数组的支持有限。例如，不能在 MergeTree 表中存储多维数组。

可以使用array函数来创建数组：

array(T)

也可以使用方括号：

[]

创建数组案例：

:) SELECT array(1, 2) AS x, toTypeName(x)

SELECT

[1, 2] AS x,

toTypeName(x)

┌─x─────┬─toTypeName(array(1, 2))─┐

│ [1,2] │ Array(UInt8) │

└───────┴─────────────────────────┘

1 rows in set. Elapsed: 0.002 sec.

:) SELECT [1, 2] AS x, toTypeName(x)

SELECT

[1, 2] AS x,

toTypeName(x)

┌─x─────┬─toTypeName([1, 2])─┐

│ [1,2] │ Array(UInt8) │

└───────┴────────────────────┘

1 rows in set. Elapsed: 0.002 sec.

3.7 元组

**Tuple(T1, T2, ...)：**元组，其中每个元素都有单独的类型。

创建元组的示例：

:) SELECT tuple(1,'a') AS x, toTypeName(x)

SELECT

(1, 'a') AS x,

toTypeName(x)

┌─x───────┬─toTypeName(tuple(1, 'a'))─┐

│ (1,'a') │ Tuple(UInt8, String) │

└─────────┴───────────────────────────┘

1 rows in set. Elapsed: 0.021 sec.

3.8 Date

日期类型，用两个字节存储，表示从 1970-01-01 (无符号) 到当前的日期值。

还有很多数据结构，可以参考官方文档：<https://clickhouse.yandex/docs/zh/data_types/>

第4章 表引擎

表引擎（即表的类型）决定了：

1）数据的存储方式和位置，写到哪里以及从哪里读取数据

2）支持哪些查询以及如何支持。

3）并发数据访问。

4）索引的使用（如果存在）。

5）是否可以执行多线程请求。

6）数据复制参数。

ClickHouse的表引擎有很多，下面介绍其中几种，对其他引擎有兴趣的可以去查阅官方文档：https://clickhouse.yandex/docs/zh/operations/table\_engines/

4.1 TinyLog

最简单的表引擎，用于将数据存储在磁盘上。每列都存储在单独的压缩文件中，写入时，数据将附加到文件末尾。

该引擎没有并发控制

- 如果同时从表中读取和写入数据，则读取操作将抛出异常；

- 如果同时写入多个查询中的表，则数据将被破坏。

这种表引擎的典型用法是 write-once：首先只写入一次数据，然后根据需要多次读取。此引擎适用于相对较小的表（建议最多1,000,000行）。如果有许多小表，则使用此表引擎是适合的，因为它比需要打开的文件更少。当拥有大量小表时，可能会导致性能低下。 不支持索引。

案例：创建一个TinyLog引擎的表并插入一条数据

:)create table t (a UInt16, b String) ENGINE=TinyLog;

:)insert into t (a, b) values (1, 'abc');

此时我们到保存数据的目录/var/lib/clickhouse/data/default/t中可以看到如下目录结构：

[root@hadoop102 t]# ls

a.bin b.bin sizes.json

a.bin 和 b.bin 是压缩过的对应的列的数据， sizes.json 中记录了每个 \*.bin 文件的大小：

[root@hadoop102 t]# cat sizes.json

{"yandex":{"a%2Ebin":{"size":"28"},"b%2Ebin":{"size":"30"}}}

4.2 Memory

内存引擎，数据以未压缩的原始形式直接保存在内存当中，服务器重启数据就会消失。读写操作不会相互阻塞，不支持索引。简单查询下有非常非常高的性能表现（超过10G/s）。

一般用到它的地方不多，除了用来测试，就是在需要非常高的性能，同时数据量又不太大（上限大概 1 亿行）的场景。

4.3 Merge

Merge 引擎 (不要跟 MergeTree 引擎混淆) 本身不存储数据，但可用于同时从任意多个其他的表中读取数据。 读是自动并行的，不支持写入。读取时，那些被真正读取到数据的表的索引（如果有的话）会被使用。

Merge 引擎的参数：一个数据库名和一个用于匹配表名的正则表达式。

案例：先建t1，t2，t3三个表，然后用 Merge 引擎的 t 表再把它们链接起来。

:)create table t1 (id UInt16, name String) ENGINE=TinyLog;

:)create table t2 (id UInt16, name String) ENGINE=TinyLog;

:)create table t3 (id UInt16, name String) ENGINE=TinyLog;

:)insert into t1(id, name) values (1, 'first');

:)insert into t2(id, name) values (2, 'second');

:)insert into t3(id, name) values (3, 'i am in t3');

:)create table t (id UInt16, name String) ENGINE=Merge(currentDatabase(), '^t');

:) select \* from t;

┌─id─┬─name─┐

│ 2 │ second │

└────┴──────┘

┌─id─┬─name──┐

│ 1 │ first │

└────┴───────┘

┌─id─┬─name───────┐

│ 3 │ i am in t3 │

└────┴────────────┘

4.4 MergeTree

Clickhouse 中最强大的表引擎当属 MergeTree （合并树）引擎及该系列（\*MergeTree）中的其他引擎。

MergeTree 引擎系列的基本理念如下。当你有巨量数据要插入到表中，你要高效地一批批写入数据片段，并希望这些数据片段在后台按照一定规则合并。相比在插入时不断修改（重写）数据进存储，这种策略会高效很多。

格式：

ENGINE [=] MergeTree(date-column [, sampling\_expression], (primary, key), index\_granularity)

参数解读：

date-column — 类型为 Date 的列名。ClickHouse 会自动依据这个列按月创建分区。分区名格式为 "YYYYMM" 。

sampling\_expression — 采样表达式。

(primary, key) — 主键。类型为Tuple()

index\_granularity — 索引粒度。即索引中相邻”标记”间的数据行数。设为 8192 可以适用大部分场景。

案例：

create table mt\_table (date Date, id UInt8, name String) ENGINE=MergeTree(date, (id, name), 8192);

insert into mt\_table values ('2019-05-01', 1, 'zhangsan');

insert into mt\_table values ('2019-06-01', 2, 'lisi');

insert into mt\_table values ('2019-05-03', 3, 'wangwu');

在/var/lib/clickhouse/data/default/mt\_tree下可以看到：

[root@hadoop102 mt\_table]# ls

20190501\_20190501\_2\_2\_0 20190503\_20190503\_6\_6\_0 20190601\_20190601\_4\_4\_0 detached

随便进入一个目录：

[root@hadoop102 20190601\_20190601\_4\_4\_0]# ls

checksums.txt columns.txt date.bin date.mrk id.bin id.mrk name.bin name.mrk primary.idx

- \*.bin是按列保存数据的文件

- \*.mrk保存块偏移量

- primary.idx保存主键索引

4.5 ReplacingMergeTree

这个引擎是在 MergeTree 的基础上，添加了“处理重复数据”的功能，该引擎和MergeTree的不同之处在于它会删除具有相同主键的重复项。数据的去重只会在合并的过程中出现。合并会在未知的时间在后台进行，所以你无法预先作出计划。有一些数据可能仍未被处理。因此，ReplacingMergeTree 适用于在后台清除重复的数据以节省空间，但是它不保证没有重复的数据出现。

格式：

ENGINE [=] ReplacingMergeTree(date-column [, sampling\_expression], (primary, key), index\_granularity, [ver])

可以看出他比MergeTree只多了一个ver，这个ver指代版本列。

案例：

create table rmt\_table (date Date, id UInt8, name String,point UInt8) ENGINE= ReplacingMergeTree(date, (id, name), 8192,point);

插入一些数据：

insert into rmt\_table values ('2019-07-10', 1, 'a', 20);

insert into rmt\_table values ('2019-07-10', 1, 'a', 30);

insert into rmt\_table values ('2019-07-11', 1, 'a', 20);

insert into rmt\_table values ('2019-07-11', 1, 'a', 30);

insert into rmt\_table values ('2019-07-11', 1, 'a', 10);

等待一段时间或optimize table rmt\_table手动触发merge，后查询

:) select \* from rmt\_table;

┌───────date─┬─id─┬─name─┬─point─┐

│ 2019-07-11 │ 1 │ a │ 30 │

└────────────┴────┴──────┴───────┘

4.6 SummingMergeTree

该引擎继承自 MergeTree。区别在于，当合并 SummingMergeTree 表的数据片段时，ClickHouse 会把所有具有相同主键的行合并为一行，该行包含了被合并的行中具有数值数据类型的列的汇总值。如果主键的组合方式使得单个键值对应于大量的行，则可以显著的减少存储空间并加快数据查询的速度，对于不可加的列，会取一个最先出现的值。

语法：

ENGINE [=] SummingMergeTree(date-column [, sampling\_expression], (primary, key), index\_granularity, [columns])

columns — 包含将要被汇总的列的列名的元组

案例：

create table smt\_table (date Date, name String, a UInt16, b UInt16) ENGINE=SummingMergeTree(date, (date, name), 8192, (a))

插入数据：

insert into smt\_table (date, name, a, b) values ('2019-07-10', 'a', 1, 2);

insert into smt\_table (date, name, a, b) values ('2019-07-10', 'b', 2, 1);

insert into smt\_table (date, name, a, b) values ('2019-07-11', 'b', 3, 8);

insert into smt\_table (date, name, a, b) values ('2019-07-11', 'b', 3, 8);

insert into smt\_table (date, name, a, b) values ('2019-07-11', 'a', 3, 1);

insert into smt\_table (date, name, a, b) values ('2019-07-12', 'c', 1, 3);

等待一段时间或optimize table smt\_table手动触发merge，后查询

:) select \* from smt\_table

┌───────date─┬─name─┬─a─┬─b─┐

│ 2019-07-10 │ a │ 1 │ 2 │

│ 2019-07-10 │ b │ 2 │ 1 │

│ 2019-07-11 │ a │ 3 │ 1 │

│ 2019-07-11 │ b │ 6 │ 8 │

│ 2019-07-12 │ c │ 1 │ 3 │

└────────────┴──────┴───┴───┘

发现2019-07-11，b的a列合并相加了，b列取了8（因为b列为8的数据最先插入）。

4.7 Distributed

分布式引擎，本身不存储数据, 但可以在多个服务器上进行分布式查询。 读是自动并行的。读取时，远程服务器表的索引（如果有的话）会被使用。

Distributed(cluster\_name, database, table [, sharding\_key])

参数解析：

cluster\_name - 服务器配置文件中的集群名,在/etc/metrika.xml中配置的

database – 数据库名

table – 表名

sharding\_key – 数据分片键

案例演示：

1）在hadoop102，hadoop103，hadoop104上分别创建一个表t

:)create table t(id UInt16, name String) ENGINE=TinyLog;

2）在三台机器的t表中插入一些数据

:)insert into t(id, name) values (1, 'zhangsan');

:)insert into t(id, name) values (2, 'lisi');

3）在hadoop102上创建分布式表

:)create table dis\_table(id UInt16, name String) ENGINE=Distributed(perftest\_3shards\_1replicas, default, t, id);

4）往dis\_table中插入数据

:) insert into dis\_table select \* from t

5）查看数据量

:) select count() from dis\_table

FROM dis\_table

┌─count()─┐

│ 8 │

└─────────┘

:) select count() from t

SELECT count()

FROM t

┌─count()─┐

│ 3 │

└─────────┘

可以看到每个节点大约有1/3的数据

第5章 SQL语法

5.1 CREATE

5.1.1 CREATE DATABASE

用于创建指定名称的数据库，语法如下：

CREATE DATABASE [IF NOT EXISTS] db\_name

如果查询中存在IF NOT EXISTS，则当数据库已经存在时，该查询不会返回任何错误。

:) create database test;

Ok.

0 rows in set. Elapsed: 0.018 sec.

5.1.2 CREATE TABLE

对于创建表，语法如下：

CREATE TABLE [IF NOT EXISTS] [db.]table\_name [ON CLUSTER cluster]

(

name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],

name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],

...

) ENGINE = engine

DEFAULT expr – 默认值，用法与SQL类似。

MATERIALIZED expr – 物化表达式，被该表达式指定的列不能被INSERT，因为它总是被计算出来的。 对于INSERT而言，不需要考虑这些列。 另外，在SELECT查询中如果包含星号，此列不会被查询。

ALIAS expr – 别名。

有三种方式创建表：

1）直接创建

:) create table t1(id UInt16,name String) engine=TinyLog

2）创建一个与其他表具有相同结构的表

CREATE TABLE [IF NOT EXISTS] [db.]table\_name AS [db2.]name2 [ENGINE = engine]

可以对其指定不同的表引擎声明。如果没有表引擎声明，则创建的表将与db2.name2使用相同的表引擎。

:) create table t2 as t1 engine=Memory

:) desc t2

DESCRIBE TABLE t2

┌─name─┬─type───┬─default\_type─┬─default\_expression─┐

│ id │ UInt16 │ │ │

│ name │ String │ │ │

└──────┴────────┴──────────────┴────────────────────┘

3）使用指定的引擎创建一个与SELECT子句的结果具有相同结构的表，并使用SELECT子句的结果填充它。

语法：

CREATE TABLE [IF NOT EXISTS] [db.]table\_name ENGINE = engine AS SELECT ...

实例：

先在t2中插入几条数据

:) insert into t1 values(1,'zhangsan'),(2,'lisi'),(3,'wangwu')

:) create table t3 engine=TinyLog as select \* from t1

:) select \* from t3

┌─id─┬─name─────┐

│ 1 │ zhangsan │

│ 2 │ lisi │

│ 3 │ wangwu │

└────┴──────────┘

5.2 INSERT INTO

主要用于向表中添加数据，基本格式如下：

INSERT INTO [db.]table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22, v23), ...

实例：

:) insert into t1 values(1,'zhangsan'),(2,'lisi'),(3,'wangwu')

还可以使用select来写入数据：

INSERT INTO [db.]table [(c1, c2, c3)] SELECT ...

实例：

:) insert into t2 select \* from t3

:) select \* from t2

┌─id─┬─name─────┐

│ 1 │ zhangsan │

│ 2 │ lisi │

│ 3 │ wangwu │

└────┴──────────┘

ClickHouse不支持的修改数据的查询：UPDATE, DELETE, REPLACE, MERGE, UPSERT, INSERT UPDATE。

5.3 ALTER

ALTER只支持MergeTree系列，Merge和Distributed引擎的表，基本语法：

ALTER TABLE [db].name [ON CLUSTER cluster] ADD|DROP|MODIFY COLUMN ...

参数解析：

ADD COLUMN – 向表中添加新列

DROP COLUMN – 在表中删除列

MODIFY COLUMN – 更改列的类型

案例演示：

1）创建一个MergerTree引擎的表

create table mt\_table (date Date, id UInt8, name String) ENGINE=MergeTree(date, (id, name), 8192);

2）向表中插入一些值

insert into mt\_table values ('2019-05-01', 1, 'zhangsan');

insert into mt\_table values ('2019-06-01', 2, 'lisi');

insert into mt\_table values ('2019-05-03', 3, 'wangwu');

3）在末尾添加一个新列age

:)alter table mt\_table add column age UInt8

:)desc mt\_table

┌─name─┬─type───┬─default\_type─┬─default\_expression─┐

│ date │ Date │ │ │

│ id │ UInt8 │ │ │

│ name │ String │ │ │

│ age │ UInt8 │ │ │

└──────┴────────┴──────────────┴────────────────────┘

:) select \* from mt\_table

┌───────date─┬─id─┬─name─┬─age─┐

│ 2019-06-01 │ 2 │ lisi │ 0 │

└────────────┴────┴──────┴─────┘

┌───────date─┬─id─┬─name─────┬─age─┐

│ 2019-05-01 │ 1 │ zhangsan │ 0 │

│ 2019-05-03 │ 3 │ wangwu │ 0 │

└────────────┴────┴──────────┴─────┘

4）更改age列的类型

:)alter table mt\_table modify column age UInt16

:)desc mt\_table

┌─name─┬─type───┬─default\_type─┬─default\_expression─┐

│ date │ Date │ │ │

│ id │ UInt8 │ │ │

│ name │ String │ │ │

│ age │ UInt16 │ │ │

└──────┴────────┴──────────────┴────────────────────┘

5）删除刚才创建的age列

:)alter table mt\_table drop column age

:)desc mt\_table

┌─name─┬─type───┬─default\_type─┬─default\_expression─┐

│ date │ Date │ │ │

│ id │ UInt8 │ │ │

│ name │ String │ │ │

└──────┴────────┴──────────────┴────────────────────┘

5.4 DESCRIBE TABLE

查看表结构

:)desc mt\_table

┌─name─┬─type───┬─default\_type─┬─default\_expression─┐

│ date │ Date │ │ │

│ id │ UInt8 │ │ │

│ name │ String │ │ │

└──────┴────────┴──────────────┴────────────────────┘

5.5 CHECK TABLE

检查表中的数据是否损坏，他会返回两种结果：

0 – 数据已损坏

1 – 数据完整

该命令只支持Log，TinyLog和StripeLog引擎。