# clickhouse

# 概述

## clickhouse定义

clickhouse是一款MPP架构，列存储的数据库管理系统（DBMS），而不只是一个数据库。主要用于在线联机分析处理（OLAP），能够使用SQL查询实时生成分析数据报告。

## 特点

1.完备的DBMS功能

DDL ( 数据定义语言 )：可以动态地创建、修改或删除数据库、表和视图，而无须重启服务。

DML ( 数据操作语言 )：可以动态查询、插入、修改或删除数据。

权限控制：可以按照用户粒度设置数据库或者表的操作权限，保障数据的安全性。

数据备份与恢复：提供了数据备份导出与导入恢复机制，满足生产环境的要求。

分布式管理：提供集群模式，能够自动管理多个数据库节点。

2.列式存储与数据压缩

3.向量化执行引擎

SIMD即****"single instruction, multiple data"（单指令流多数据流）****，是Flynn分类法对计算机的四大分类之一。****它本质上是采用一个控制器来控制多个处理器，同时对一组数据中的每一条分别执行相同的操作，从而实现空间上的并行性的技术。****

4.关系模型和SQL查询：

相比HBase和Redis这类NoSQL数据库，ClickHouse使用关系模型描述数据并提供了传统数据库的概念 ( 数据库、表、视图和函数等 )。与此同时，ClickHouse完全使用SQL作为查询语言 ( 支持GROUP BY、ORDER BY、JOIN、IN等大部分标准SQL )，这使得它平易近人，容易理解和学习。

5.多样化表引擎

6.分布式和多线程

多线程：如果说向量化执行是通过数据级并行的方式提升了性能，那么多线程处理就是通过线程级并行的方式实现了性能的提升。相比基于底层硬件实现的向量化执行SIMD，线程级并行通常由更高层次的软件层面控制。

多核并行：MySQL单条SQL是单线程的，只能跑满一个core，ClickHouse相反，ClickHouse将数据划分为多个partition，每个partition再进一步划分为多个index granularity，然后通过多个CPU核心分别处理其中的一部分来实现并行数据处理。在这种设计下，单条Query就能利用整机所有CPU。极致的并行处理能力，极大的降低了查询延时。

分布式：分布式就是使用多台机器。 ClickHouse会自动将查询拆解为多个task下发到集群中

7.ClickHouse在数据存取方面，既支持分区 ( 纵向扩展，利用多线程原理 )，也支持分片 ( 横向扩展，利用分布式原理 )，可以说是将多线程和分布式的技术应用到了极致。

8.多主架构

主从架构是一个master管理全局

多主架构的每个节点都是对等的，客户端访问任意一个节点都能得到相同的效果，天然规避了单点故障问题，非常适合用于多数据中心、异地多活的场景。

9.在线查询：查询速度很快，可以很快返回结果

数据分片和分布式查询

10.有序存储：在建表时就可以指定根据某列进行排序

11.支持主键：主键查询非常快

12.数据存储于磁盘：许多面向列的DBMS（SAP HANA和GooglePowerDrill）只能在内存中工作。但即使在数千台服务器上，内存也太小，无法在Yandex.Metrica中存储所有浏览量和会话。

13.实时数据更新：ClickHouse支持主键表。为了快速执行对主键范围的查询，数据使用合并树(MergeTree)进行递增排序。由于这个原因，数据可以不断地添加到表中。添加数据时无锁处理。

14.支持近似计算：

（1）系统包含用于近似计算各种值，中位数和分位数的集合函数。

（2）支持基于部分（样本）数据运行查询并获得近似结果。

（3）支持为有限数量的随机密钥（而不是所有密钥）运行聚合。在数据中密钥分发的特定条件下，这提供了相对准确的结果，同时使用较少的资源。

clickhouse缺点：

不支持事务

不支持窗口函数

元数据管理需要人工干预维护

支持有限的操作系统：现在支持ubuntu,centos 需要自己编译，不过有热心人已经编译好了，拿来用就行。对于Windows 不支持。

## 适合使用的场景

做olap分析

读多于写：适合查询数据，不适合单行更新

写数据需要大批次（大于1000行）进行，

高吞吐写入能： 能够达到50MB-200MB/s的写入吞吐能力，按照每行100Byte估算，大约相当于50W-200W条/s的写入速度

大宽表，包含大量的列

不需要事务，数据一致性要求较低

查询结果显著小于数据源。即数据有过滤和聚合，返回结果不超过单个服务器内存大小

## 为什么clickhouse查询速度快？

1. 列式存储，减少数据扫描范围
2. 数据压缩，减少数据传输大小

数据总体的压缩比可以达到8:1

1. 向量化执行引擎：

为了实现向量化执行，需要利用CPU的SIMD指令。SIMD的全称是Single Instruction Multiple Data，即用单条指令操作多条数据，可以通过一条CPU指令对一组数据执行相同的操作，实现空间上的并行。现代计算机系统概念中，它是通过数据并行以提高性能的一种实现方式 ( 其他的还有指令级并行和线程级并行 )，它的原理是在CPU寄存器层面实现数据的并行操作。

1. 多线程
2. 分布式
3. 支持主键：主键查询非常快
4. 支持近似计算
5. 支持索引（一级、二级、稀疏索引）
6. C++ 写的，更好的调用硬件资源
7. 摒弃了Hadoop框架
8. 支持预计算：例如sumingmergetree引擎

## ClickHouse 和一些技术的比较

商业OLAP数据库

例如：HP Vertica, Actian the Vector,

区别：ClickHouse是开源而且免费的

云解决方案

例如：亚马逊RedShift和谷歌的BigQuery

区别：ClickHouse可以使用自己机器部署，无需为云付费

Hadoop生态软件

例如：Cloudera Impala, Spark SQL, Facebook Presto , Apache Drill

区别：

ClickHouse支持实时的高并发系统

ClickHouse不依赖于Hadoop生态软件和基础

ClickHouse支持分布式机房的部署

开源OLAP数据库

例如：InfiniDB, MonetDB, LucidDB

区别：这些项目的应用的规模较小，并没有应用在大型的互联网服务当中，相比之下，ClickHouse的成熟度和稳定性远远超过这些软件。

开源分析，非关系型数据库

例如：Druid , Apache Kylin

区别：ClickHouse可以支持从原始数据的直接查询，ClickHouse支持类SQL语言，提供了传统关系型数据的便利。

# 安装

## 安装前准备

1.验证操作系统

1.1. 查看当前linux版本

查看OS版本： uname -a

Linux uat-db-28-116 3.10.0-514.el7.x86\_64 #1 SMP Wed Oct 19 11:24:13 EDT 2016 x86\_64 x86\_64 x86\_64 GNU/Linux

$: cat /proc/version

Linux version 2.6.32-573.el6.x86\_64 (mockbuild@c6b9.bsys.dev.centos.org) (gcc version 4.4.7 20120313 (Red Hat 4.4.7-16) (GCC) ) #1 SMP Thu Jul 23 15:44:03 UTC 2015

1.2检查系统是否支持SSE 4.2：

grep -q sse4\_2 /proc/cpuinfo && echo "SSE 4.2 supported" || echo "SSE 4.2 not supported"

2.禁用大页(Huge Pages)

始终禁用透明大页(transparent huge pages)。 它会干扰内存分配器，从而导致显着的性能下降。

echo 'never' | sudo tee /sys/kernel/mm/transparent\_hugepage/enabled

1.3、禁用交换内存：

echo "vm.swappiness = 1" >> /etc/sysctl.conf

echo 1 > /proc/sys/vm/swappiness

1.4、关闭防火墙

sudo systemctl status firewalld

sudo systemctl stop firewalld

sudo systemctl disable firewalld

sudo systemctl status firewalld

CentOS取消SELINUX安全检查

修改/etc/selinux/config中的SELINUX=disabled

1.5、取消文件限制

vim /etc/security/limits.conf

vim /etc/security/limits.d/20-nproc.conf

文件末尾追加

\* soft nofile 65536

\* hard nofile 65536

\* soft nproc 131072

\* hard nproc 131072

安装依赖（可不安）

sudo yum install -y libtool

sudo yum install -y \*unixODBC\*

该依赖与clickhouse的ODBC表引擎有关

网络要求

如果可能的话，使用10G或更高级别的网络。

网络带宽对于处理具有大量中间结果数据的分布式查询至关重要。 此外，网络速度会影响复制过程。

## 在线安装

1. 先确保安装了curl，没有的话，安装一下

sudo yum install -y curl

1. 添加clickhouse源

curl -s https://packagecloud.io/install/repositories/altinity/clickhouse/script.rpm.sh | sudo bash

由于网络原因，可能会失败，多试几次

1. 查一下clickhouse源有没有添加成功

sudo yum list 'clickhouse\*'

1. 安装servier和client

sudo yum install -y clickhouse-server clickhouse-client

1. 检查是否安装成功

sudo yum list installed 'clickhouse\*'

## 离线安装

下载对应的linux版本的clickhouse rpm版本：

clickhouse-client-21.4.6.55-2.noarch.rpm

clickhouse-server-21.4.6.55-2.noarch.rpm

clickhouse-common-static-21.4.6.55-2.x86\_64.rpm

官网下载地址：https://repo.yandex.ru/clickhouse/rpm/stable/x86\_64/

通过rpm方式按下面顺序依次安装：

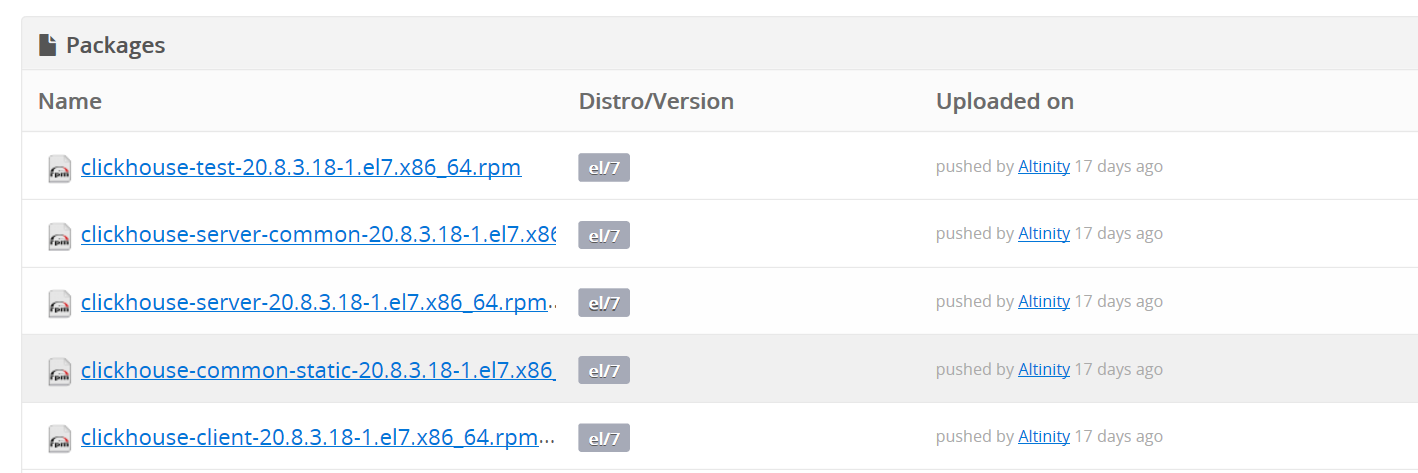
进入rpm所在目录：/oradata/cksoft/

rpm -ivh clickhouse-\*.rpm

另一种方式

下载地址：<https://packagecloud.io/altinity/clickhouse>

一般是5个安装包



以如下顺序，rpm方式安装

rpm -ivh clickhouse-common-static-20.3.11.97-1.el7.x86\_64.rpm  
rpm -ivh clickhouse-server-common-20.3.11.97-1.el7.x86\_64.rpm  
rpm -ivh clickhouse-server-20.3.11.97-1.el7.x86\_64.rpm  
rpm -ivh clickhouse-client-20.3.11.97-1.el7.x86\_64.rpm

创建目录

mkdir -p /app/clickhouse/log  
mkdir -p /app/clickhouse/data  
mkdir -p /app/clickhouse/data\_old  
mkdir -p /app/clickhouse/tmp/  
mkdir -p /app/clickhouse/format\_schemas/  
mkdir -p /app/clickhouse/user\_files/

## 文件目录

### 服务端目录

$: ll /etc/clickhouse-server/

-rw-r--r-- 1 root root 17642 config.xml

-rw-r--r-- 1 root root 5609 users.xml

### 客户端目录

$: ll /etc/clickhouse-client/

drwxr-xr-x 2 clickhouse clickhouse 4096 conf.d

-rw-r--r-- 1 clickhouse clickhouse 1568 config.xml

### 日志目录

ll /var/log/clickhouse-server/

-rw-r-----. 1 clickhouse clickhouse 20629 May 25 23:34 clickhouse-server.err.log

-rw-r-----. 1 clickhouse clickhouse 7530125 May 26 09:28 clickhouse-server.log

-rw-r-----. 1 clickhouse clickhouse 753 May 25 23:26 stderr.log

-rw-r-----. 1 clickhouse clickhouse 0 May 25 23:14 stdout.log

### 数据目录

[root@vm00 testdata]# ll /var/lib/clickhouse/

total 4

drwxr-x---. 2 clickhouse clickhouse 116 May 25 23:14 access

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 cores

drwxr-x---. 5 clickhouse clickhouse 50 May 25 23:16 data

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 dictionaries\_lib

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 flags

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 format\_schemas

drwxr-x---. 4 clickhouse clickhouse 106 May 25 23:16 metadata

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 metadata\_dropped

drwxr-x---. 2 clickhouse clickhouse 41 May 25 23:14 preprocessed\_configs

-rw-r-----. 1 clickhouse clickhouse 58 May 25 23:14 status

drwxr-x---. 8 clickhouse clickhouse 72 May 25 23:15 store

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 tmp

drwxr-x---. 2 clickhouse clickhouse 6 May 25 23:14 user\_files

#### data目录详解

data存放表数据

1.partition 分区目录

partition\_n目录下的各类数据文件都是以分区形式被组织存放的，属于相同分区的数据最终会被合并到同一个分区目录内。

2.checksums.txt：校验文件，使用二进制存储，保存了各类文件的size大小和size的哈希值，用于快速校验文件的完整性和正确性。

3.columns.txt：列信息文件，使用文本文件存储，用于保存分区下的列字段信息。

4.count.txt：计数文件，文本文件存储，用于记录当前数据分区目录下数据的总行数。

5.primary.idx：以及索引文件，使用二进制格式存储。用于存放稀疏索引，一张MergeTree表只能声明一次一级索引

(通过order by或者primary key)。借助稀疏索引在数据查询的时候能够排除主键范围之外的数据文件，从而减少数据扫描范围，加速查询速度。

6.[column].bin：数据文件，使用压缩格式存储，默认使用LZ4压缩格式，用于存储某一列的数据。由于MergeTree采用列式存储，每个列字段都拥有独立的bin数据文件，并以列字段命名。

7.[column].mrk列字段标记，使用二进制格式存储。标记文件中保存了bin文件中数据的偏移量信息，标记文件与稀疏文件对齐，又与bin文件一一对应，所以MergeTree通过标记文件建立了primary.idx稀疏索引与bin数据文件的隐射关系。

首先通过primary.idx找到对应数据的偏移量信息(.mrk),再通过偏移量直接从bin文件中读取数据。由于.mrk标记文件与.bin文件一一对应，所以MergeTree中的每个列字段都会拥有与其对应的.mrk文件。

8.[column].mrk2 如使用了自适应大小的索引间隔，则标记文件会以.mrk2命名。工作原理和作用和.mrk标记文件相同。

9.partition.dat和minmax\_[column].idx：

若使用了分区键则会额外生成partition.dat和minmax索引文件，均使用二进制格式存储。partition.dat用于保存当前分区下

分区表达式最终生成值，minmax索引文件用于记录当前分区字段对应原始数据的最小值和最大值。

在分区索引作用下，进行数据查询时候能够快速跳过不必要的数据分区目录，从而减少最终需要扫描的数据范围。

10.skp\_idx\_[column].idx和skp\_idx\_[column].mrk：

若在建表语句中声明了二级索引则会额外生成相应的二级索引与标记文件，他们同样用二进制存储。

二级索引在clickhouse中又称之为跳数索引，目前拥有minmax，set，ngrambf\_v1和tokenbf\_v1四种类型。

这些索引的目标和一级稀疏索引相同，为了进一步减少所需要扫描的数据范围，以加速整个查询过程。

# 配置

## 修改数据存放目录

修改/etc/clickhouse-server/config.xml

默认配置为：

<!-- Path to data directory, with trailing slash. -->

<path>/var/lib/clickhouse/</path>

修改之后：

<!-- Path to data directory, with trailing slash. -->

<path>/home/clickhouse/data/clickhouse</path>

注意这个配置的目录磁盘空间必须足够大

其他配置可以根据自己的实际情况而定，注意配置端口是否被占用

## 开放外部访问

修改/etc/clickhouse-server/config.xml

<listen\_host>0.0.0.0</listen\_host>

修改/etc/clickhouse-server/config.xml

<!-- Listen specified host. use :: (wildcard IPv6 address), if you want to accept connections both with IPv4 and IPv6 from everywhere. -->

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

修改/etc/clickhouse-server/users.xml

<networks incl="networks" replace="replace">

<ip>::/0</ip>

</networks>

## 修改用户密码

### default用户

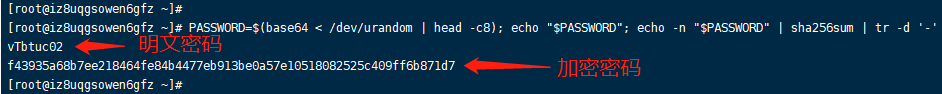
#### 明文

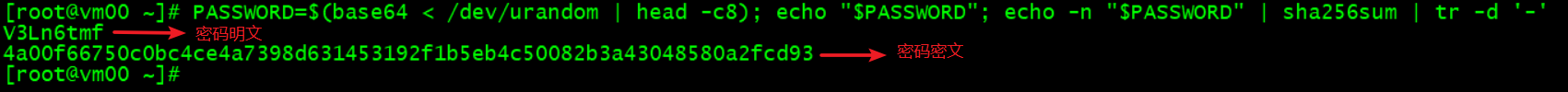
修改/etc/clickhouse-server/users.xml

<password>明文密码</password>

#### 密文

PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'





这样可以得到两行数据，第一行是密码明文，第二行是密码密文

将密文填入配置文件

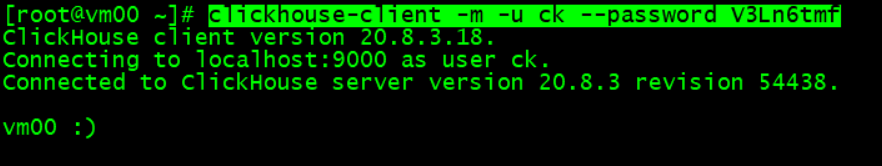
修改/etc/clickhouse-server/users.xml

找到 users --> default --> 标签下的password修改成password\_sha256\_hex，并把密文填进去

<password\_sha256\_hex>密码密文</password\_sha256\_hex>

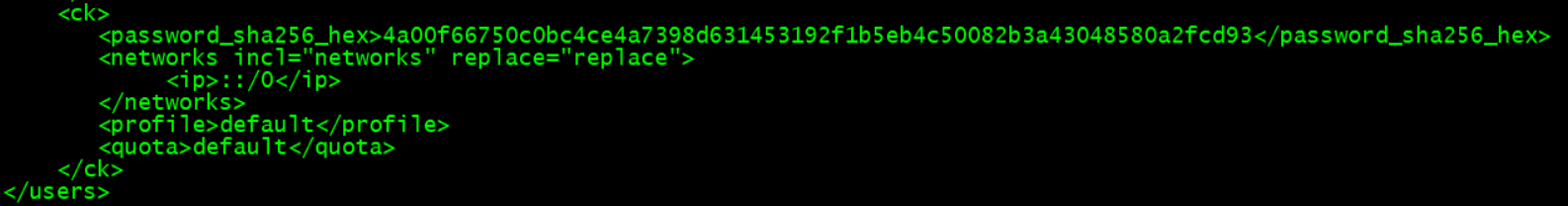
客户端连接时输入明文

clickhouse-client -h ip地址 -d default -m -u default --password 密码明文



### 普通用户





ck即为普通用户的用户名

## 集群配置

### metrika.xml

metrika.xml配置文件的路径是任意的，一般配置在/etc/clickhouse-server/目录下，我们只需将metrika.xml的路径配置到config.xml中即可。

vi /etc/clickhouse-server/metrika.xml

<clickhouse\_remote\_servers>

<!--集群名称，clickhouse支持多集群的模式-->

<clickhouse\_cluster>

<!--定义分片节点，这里我指定3个分片，每个分片只有1个副本，也就是它本身-->

<shard>

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

<replica>

<host>server1</host>

<port>9000</port>

</replica>

</shard>

<shard>

<replica>

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

<host>server2</host>

<port>9000</port>

</replica>

</shard>

<shard>

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

<replica>

<host>server3</host>

<port>9000</port>

</replica>

</shard>

</clickhouse\_cluster>

</clickhouse\_remote\_servers>

<!--zookeeper集群的连接信息-->

<zookeeper-servers>

<node index="1">

<host>server1</host>

<port>2181</port>

</node>

<node index="2">

<host>server1</host>

<port>2182</port>

</node>

<node index="3">

<host>server1</host>

<port>2183</port>

</node>

</zookeeper-servers>

<!--定义宏变量，后面需要用-->

<macros>

<replica>server1</replica>

</macros>

<!--不限制访问来源ip地址-->

<networks>

<ip>::/0</ip>

</networks>

<!--数据压缩方式，默认为lz4-->

<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>

配置好后将该配置文件分发到集群的各个节点。注意分发后同时修改宏变量macros为对应主机名。

### config.xml

<!--引入metrika.xml-->

<include\_from>/etc/clickhouse-server/config.d/metrika.xml</include\_from>

#引用Zookeeper配置的定义

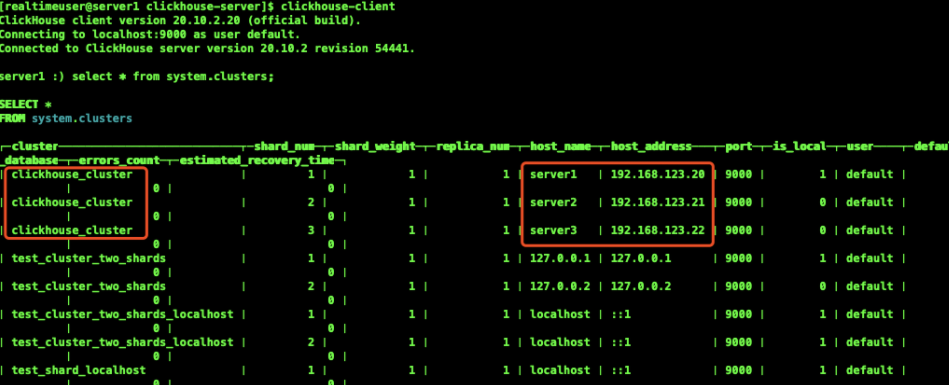
<zookeeper incl="zookeeper-servers" optional="true" />

#打开注释，让其他节点访问当前节点ClickHouse

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

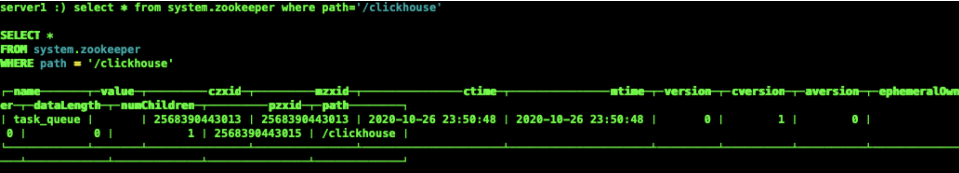
### 集群启动

先启动zookeeper，再启动每个clickhouse节点



使用ClickHouse访问Zookeeper

#查询Zookeeper根目录 select \* from system.zookeeper where path = '/' #查询ClickHouse目录 select \* from system.zookeeper where path = '/clickhouse'



### 数据副本

#### zookeeper

clickhouse集群不依赖与zookeeper，当集群需要进行副本备份时才需要zookeeper。

clickhouse依赖于zookeeper进行副本备份：一个客户端向其中一个实例发起写入数据的操作，该实例接收到请求后会将操作日志写入zookeeper中，这样其他实例监听到zookeeper的变化，便从目标实例下载数据作为副本。

要使用副本，需在配置文件中设置 ZooKeeper 集群的地址：

<zookeeper>

<node index="1">

<host>example1</host>

<port>2181</port>

</node>

<node index="2">

<host>example2</host>

<port>2181</port>

</node>

<node index="3">

<host>example3</host>

<port>2181</port>

</node>

</zookeeper>

需要 ZooKeeper 3.4.5 或更高版本。

你可以配置任何现有的 ZooKeeper 集群，系统会使用里面的目录来存取元数据（该目录在创建可复制表时指定）。

如果配置文件中没有设置 ZooKeeper ，则无法创建复制表，并且任何现有的复制表都将变为只读。

SELECT 查询并不需要借助 ZooKeeper ，副本并不影响 SELECT 的性能，查询复制表与非复制表速度是一样的。查询分布式表时，ClickHouse的处理方式可通过设置 [max\_replica\_delay\_for\_distributed\_queries](https://clickhouse.tech/docs/zh/operations/settings/settings/" \l "settings-max_replica_delay_for_distributed_queries) 和 [fallback\_to\_stale\_replicas\_for\_distributed\_queries](https://clickhouse.tech/docs/zh/operations/settings/settings/) 修改。

对于每个 INSERT 语句，会通过几个事务将十来个记录添加到 ZooKeeper。（确切地说，这是针对每个插入的数据块; 每个 INSERT 语句的每 max\_insert\_block\_size = 1048576 行和最后剩余的都各算作一个块。）相比非复制表，写 zk 会导致 INSERT 的延迟略长一些。但只要你按照建议每秒不超过一个 INSERT 地批量插入数据，不会有任何问题。一个 ZooKeeper 集群能给整个 ClickHouse 集群支撑协调每秒几百个 INSERT。数据插入的吞吐量（每秒的行数）可以跟不用复制的数据一样高。

# 服务启动关闭命令

查看状态：systemctl status clickhouse-server

启动：systemctl start clickhouse-server

停止：systemctl stop clickhouse-server

重启：systemctl restart clickhouse-server

关闭开机自启：systemctl disable clickhouse-server

添加开机自启：chkconfig clickhouse-server on

关闭开机自启：chkconfig clickhouse-server off

还可以使用下面一组命令：

service clickhouse-server status

service clickhouse-server start

service clickhouse-server stop

service clickhouse-server restart

# 客户端

## 命令行客户端

clickhouse-client --host xxx --port 9000 -d default -u default --password 密码明文 -m

命令行参数

--host, -h -– 服务端的host名称, 默认是localhost。您可以选择使用host名称或者IPv4或IPv6地址。

--port – 连接的端口，默认值：9000。注意HTTP接口以及TCP原生接口使用的是不同端口。

--user, -u – 用户名。 默认值：default。

--password – 密码。 默认值：空字符串。

--query, -q – 使用非交互模式查询。

--database, -d – 默认当前操作的数据库. 默认值：服务端默认的配置（默认是default）。

--multiline, -m – 如果指定，允许多行语句查询（Enter仅代表换行，不代表查询语句完结）。

--multiquery, -n – 如果指定, 允许处理用;号分隔的多个查询，只在非交互模式下生效。

--format, -f – 使用指定的默认格式输出结果。

--vertical, -E – 如果指定，默认情况下使用垂直格式输出结果。这与–format=Vertical相同。在这种格式中，每个值都在单独的行上打印，这种方式对显示宽表很有帮助。

--time, -t – 如果指定，非交互模式下会打印查询执行的时间到stderr中。

--stacktrace – 如果指定，如果出现异常，会打印堆栈跟踪信息。

--config-file – 配置文件的名称。

--secure – 如果指定，将通过安全连接连接到服务器。

--param\_<name> — 查询参数配置[查询参数](https://clickhouse.tech/docs/zh/interfaces/cli/" \l "cli-queries-with-parameters)。

### 使用方式

**交互式**

通过交互式查询的语句会默认被记录到~/.clickhouse-client-history文件中，该记录可以作为审计之用。

**非交互式**非交互模式主要适用于批处理场景。如数据的导入导出等操作。

导入：

# cat t.tsv| clickhouse-client --query "insert into t from tsv"

导出：

#clickhouse-client --query "select \* from t" > t.tsv

默认情况下只能执行一条SQL，使用参数--multiquery 则可以支持一次运行多条SQL查询，

多条SQL语句之间使用分号间隔：

示例：clickhouse-client -h 192.168.8.101 --multiquery --query="select 10;select 20;select 30;"

指定--multiline，需要在末尾加分号，并且按下enter键才会执行，

也可以用\G代替分号，这是输出的结果格式为Vertical，比较适合宽表

没有指定multiline，换行需要在每行末尾加\

要使用批处理模式，请指定query参数，或将数据发送到stdin(它会验证stdin是否是终端)，或两者同时进行。与HTTP接口类似，当使用query参数并向stdin发送数据时，客户端请求就是一行一行的stdin输入作为query的参数。**这种方式在大规模的插入请求中非常方便。**

使用客户端插入数据的示例：

1. 插入单行数据：

echo -ne "1, 'some text', '2016-08-14 00:00:00'\n2, 'some more text', '2016-08-14 00:00:01'" | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

1. 开启输入窗口：

cat <<\_EOF | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

3, 'some text', '2016-08-14 00:00:00'

4, 'some more text', '2016-08-14 00:00:01'

\_EOF

1. 读取文件并插入

cat file.csv | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

## 原生TCP客户端

原生接口用于[命令行客户端](https://clickhouse.tech/docs/zh/interfaces/cli/)，用于分布式查询处理期间的服务器间通信，以及其他C++程序。

TCP协议拥有更好的性能，默认是9000端口

## HTTP客户端

HTTP接口允许您在任何编程语言的任何平台上使用ClickHouse，具有更好的兼容性，可以通过REST服务的形式而被广泛用于java Python Golang等编程语言的客户端。但是比原生接口受到更多的限制。

默认的端口为8123

## 连接工具

DbeaverEE

DataGrip

Tabix

HouseOps

# 卸载

查看已安装包

rpm -qa | grep clickhouse

卸载clickhouse相关软件

rpm -e clickhouse-client-20.5.4.40-2.noarch --nodeps

rpm -e clickhouse-server-20.5.4.40-2.noarch --nodeps

rpm -e clickhouse-common-static-20.5.4.40-2.x86\_64 --nodeps

删除相关的目录和数据

#数据目录

rm -rf /var/lib/clickhouse

#删除集群配置文件

rm -rf /etc/metrika.xml

#删除配置文件

rm -rf /etc/clickhouse-\*

#删除日志文件

rm -rf /var/log/clickhouse-server

全局查找clickhouse文件和目录，如果存在，则全部删除

find / -name clickhouse

# 查询信息的SQL

## 查询集群信息

select \* from system.clusters;

## 查询SQL执行计划

ch --send\_logs\_level=trace <<< 'select \* from tutorial.hits\_v1' > /dev/null

## 查询后台运行的查询

select

query\_id,read\_rows,total\_rows\_approx,memory\_usage,

initial\_user,initial\_address,elapsed,query

from system.processes;

## 杀死后台的查询

kill query where query\_id='70442d9b-7fc5-4a0e-81be-9543431a4882';

## 查询运行的mutation

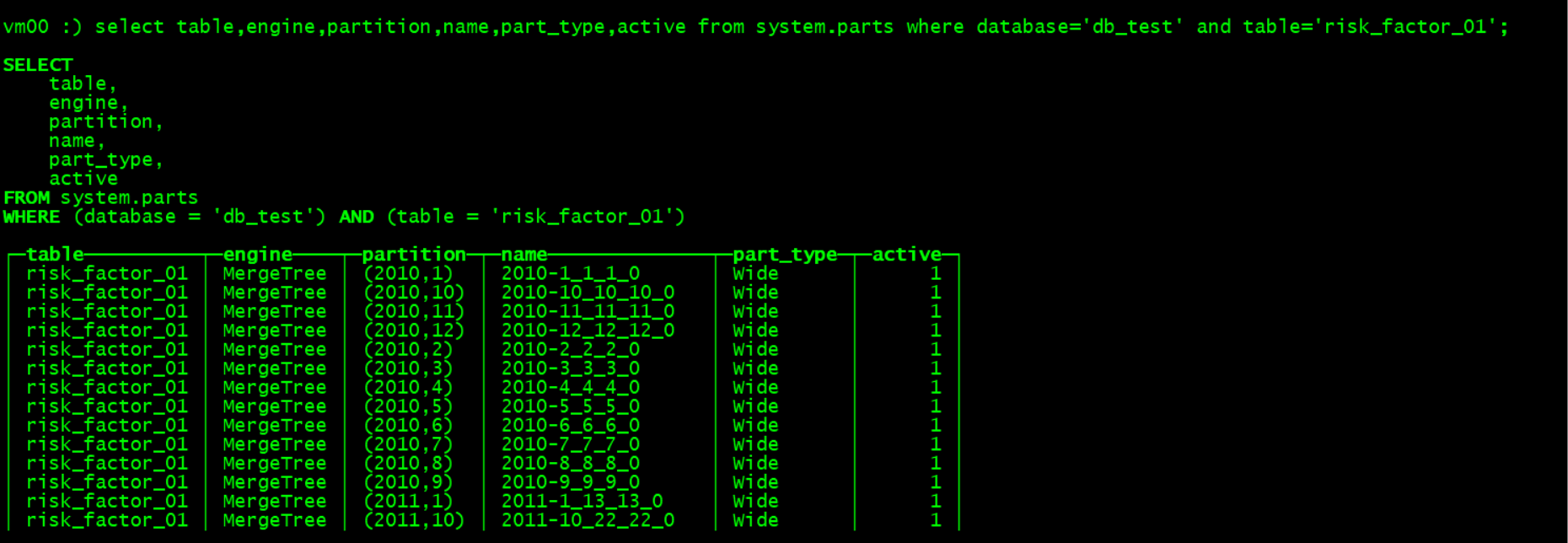
SELECT database, table, mutation\_id, command, create\_time, is\_done FROM system.mutations;

## 杀死运行的mutation

KILL MUTATION WHERE mutation\_id = ‘mutation\_id’;

## 查询分区信息

select table,engine,partition,name,part\_type,active,path from system.parts where database='db\_test' and table='risk\_factor\_01';



## 手动合并分区

optimize table tb\_name final;

对单个分区进行合并：OPTIMIZE TABLE tb\_name PARTITION 429-6001;

## 查看某字段类型

select toTypeName(字段名) from 表名;

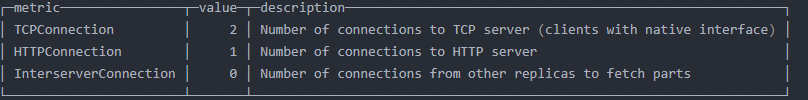
## 查看建表语句

SHOW CREATE TABLE 表名;

## 当前连接数

众所周知，CH 对外暴露的原生接口分为 TCP 和 HTTP 两类，通过 system.metrics 即可查询当前的 TCP、HTTP 与内部副本的连接数。

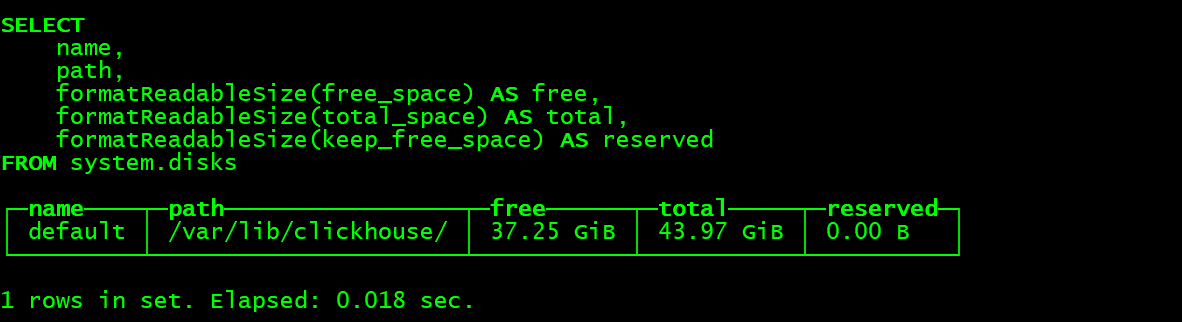
SELECT \* FROM system.metrics WHERE metric LIKE '%Connection'; SELECT \* FROM system.metrics WHERE metric LIKE '%Connection'



## 存储空间统计

查询 CH 各个存储路径的空间:

SELECT name,path,formatReadableSize(free\_space) AS free,formatReadableSize(total\_space) AS total,formatReadableSize(keep\_free\_space) AS reserved FROM system.disks



## 慢查询

查看哪些查询比较慢

SELECT

user,

client\_hostname AS host,

client\_name AS client,

formatDateTime(query\_start\_time,'%T') AS started,

query\_duration\_ms / 1000 AS sec,

round(memory\_usage / 1048576) AS MEM\_MB,

result\_rows AS RES\_CNT,

result\_bytes / 1048576 AS RES\_MB,

read\_rows AS R\_CNT,

round(read\_bytes / 1048576) AS R\_MB,

written\_rows AS W\_CNT,

round(written\_bytes / 1048576) AS W\_MB,

query

FROM system.query\_log

WHERE type = 2

ORDER BY query\_duration\_ms DESC

LIMIT 10;

## 各数据库占用空间统计

SELECT database, formatReadableSize(sum(bytes\_on\_disk)) AS on\_disk FROM system.parts GROUP BY database

## 各字段占用空间统计

SELECT

database,

table,

column,

any(type),

sum(column\_data\_compressed\_bytes) AS compressed,

sum(column\_data\_uncompressed\_bytes) AS uncompressed,

round(uncompressed / compressed, 2) AS ratio,

compressed / sum(rows) AS bpr,

sum(rows)

FROM system.parts\_columns

WHERE active AND database != 'system'

GROUP BY

database,

table,

column

ORDER BY

database ASC,

table ASC,

column ASC

## 查看库表资源占用情况

select database,

table,

sum(rows) AS "总行数",

formatReadableSize(sum(data\_uncompressed\_bytes)) as "原始大小",

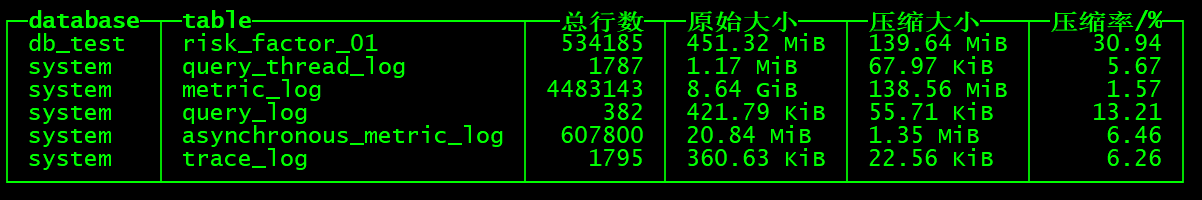
formatReadableSize(sum(data\_compressed\_bytes)) AS "压缩大小",

round((sum(data\_compressed\_bytes) / sum(data\_uncompressed\_bytes)) \* 100.,2) AS "压缩率/%"

from system.parts

group by database,table

order by database;



## 副本预警监控

SELECT database, table, is\_leader, total\_replicas, active\_replicas

FROM system.replicas

WHERE is\_readonly

OR is\_session\_expired

OR future\_parts > 30

OR parts\_to\_check > 20

OR queue\_size > 30

OR inserts\_in\_queue > 20

OR log\_max\_index - log\_pointer > 20

OR total\_replicas < 2

OR active\_replicas < total\_replicas

# 数据类型

## 整型

### 有符号整型

范围：-2^(n-1)~2^(n-1)-1

Int8：-128~127

Int16：-32768~32767

Int32：-2147483648~2147483647

Int64：-922337203854775808~922337203854775807

### 无符号整型

范围：0~2^n-1

UInt8：0~255

UInt16：0~65535

UInt32：0~4294967295

UInt64：0~18446744073709551615

## 浮点型



clickhouse 直接使用Float32代表单精度浮点数 使用Float64表示双精度浮点数。

使用浮点数需要注意它的精度是有限的，Float32从小数点后第8位，Float64从小数点后的第17位起会产生数据溢出。

建议尽可能使用整型存储数据，浮点数会引起误差！

正无穷：inf

负无穷: -inf

非数字: nan

## Decimal

若需要要求更高的精度的数值运算，则需要使用定点数。Clickhouse提供了Decimal32，Decimal64，Decimal128三种精度的定点数。简写为Decimal32(S),Decimal64(S),Decimal128(S),原生方式为Decimal（P,S）：

其中P表示精度precision，决定总位数（整数部分+小数位部分），取值范围为1--38

S代表规模 scale，决定小数位，取值范围是0--P。

简写方式和原生方式的对应表：

Decimal类型：



四则运算规则：



## 字符串类型

### String

String表示字符串类型，长度不限。在声明String类型的时候无须声明字符串的大小。String类型不限定字符集，理论上可以将任意编码的字符串存进来。但是为了规范和统一，推荐使用UTF-8字符集。

### FixedString

和MySQL的CHAR比较类似，对于有明确长度的字符串可以使用。

和char不同的是FixedString使用Null字节填充末尾字符，而char使用空格填充。

## UUID

通用唯一标识符（UUID）是一个16字节的数字，用于标识记录

32位，格式8-4-4-4-12

UUID类型值的示例如下:

61f0c404-5cb3-11e7-907b-a6006ad3dba0

如果在插入新记录时未指定UUID列的值，则UUID值将用零填充:

00000000-0000-0000-0000-000000000000

生成UUID：

generateUUIDv4()

## 布尔类型

clickhouse不支持布尔类型，可以使用UInt8限制值为0或1

## 时间类型

### Date

Date 不包含具体的时间信息，只精确到天，支持字符串形式写入。 范围：1970-01-01 2105-12-31

### DateTime

Datetime类型包含时分秒，精确到秒，支持用字符串形式插入。 [1970-01-01 00:00:00, 2105-12-31 23:59:59] 可以配合时区使用。

### DateTime64

Datetime64 可以记录亚秒，在Datetime上增加了精度的设置。 语法：DateTime64(precision, [timezone])

### Interval

IntervalYear

IntervalQuarter

IntervalMonth

IntervalDay

IntervalHour

IntervalSecond

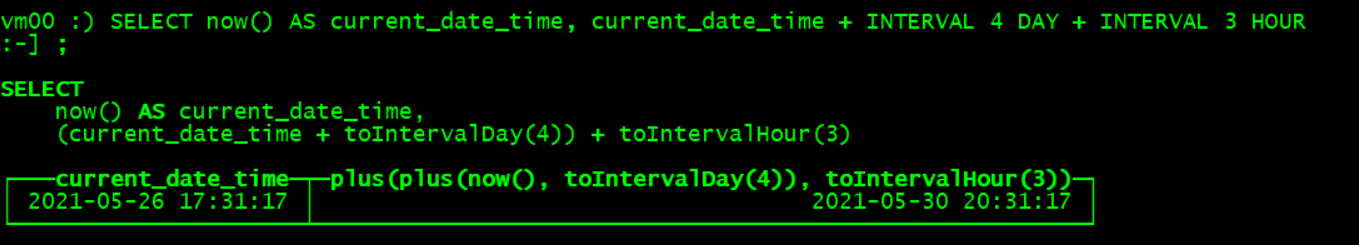
IntervalMinute

IntervalWeek

获取年份 季度 月份 日期 时 分 秒 周等信息。

用法示例：

SELECT now() AS current\_date\_time, current\_date\_time + INTERVAL 4 DAY + INTERVAL 3 HOUR



## 数组

Array（类型）

## 元组

Tuple(类型1，类型2，类型3......)

## 枚举

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。

## 嵌套类型

Nested（Name1 Type1，Name2 Type2，…）

相当于表中嵌套一张表，插入时相当于一个多维数组的格式，一个字段对应一个数组

## Nullable(type)

只能与基础数据类型搭配使用，表示某个类型的值可以为NULL

注意：  
1）不能与复合类型数据一起使用  
2）不能作为索引字段  
3）尽量避免使用，字段被Nullable修饰后会额外生成[Column].null.bin 文件保存Null值，增加开销

## Domain

IPv4 使用 UInt32 存储。如 116.253.40.133  
IPv6 使用 FixedString(16) 存储。如 2a02:aa08:e000:3100::2

# 数据库引擎

## MySQL

### 介绍

MySQL引擎用于将远程的MySQL服务器中的表映射到ClickHouse中，并允许您对表进行INSERT和SELECT查询，方便在ClickHouse与MySQL之间进行数据交换。

MySQL库引擎中,所有的表都是MySQL表引擎,通过库引擎的方式创建的表无法进行更新/删除操作,也无法进行表的DDL操作.  
表允许删除(drop)与卸载(detach),但是不会删除mysql库中的表,并且删除或者卸载之后,允许通过装载(attach)再将表装载回来.

MySQL数据库引擎会将对其的查询转换为MySQL语法并发送到MySQL服务器中，因此可以执行诸如SHOW TABLES或SHOW CREATE TABLE之类的操作。

但无法对其执行以下操作：

RENAME

CREATE TABLE

ALTER

### 建库语法

**CREATE** **DATABASE** [**IF** **NOT** **EXISTS**] db\_name [**ON** **CLUSTER** **cluster**]ENGINE = MySQL('host:port', ['database' | **database**], 'user', 'password')

### 参数详解

host:port — 链接的MySQL地址。

database — 链接的MySQL数据库。

user — 链接的MySQL用户。

password — 链接的MySQL用户密码。

## MaterializeMySQL

(目前处于试验阶段)

### 介绍

在Clickhouse 20.8.2.3 版本中新增一个数据库引擎，将clickhouse模拟为MySQL的从库，可以通过mysql的binlog实时的接收来自mysql的数据并在clickhouse物化,极大提升了数仓的查询性能和数据同步的时效性。同时增加了获取mysql数据的方式，除了mysql协议和mysql函数和mysql表引擎，clickhouse可以作为mysql的从库。

### 注意点

1.支持mysql 库级别的数据同步，暂不支持表级别的。

2.MySQL 库映射到clickhouse中自动创建为ReplacingMergeTree 引擎的表

3.支持全量和增量同步，首次创建数据库引擎时进行一次全量复制，之后通过监控binlog变化进行增量数据同步

4.支持的MySQL版本：5.6 5.7 8.0

5.支持的操作:insert，update，delete，alter，create，drop，truncate等大部分DDL操作

7.支持的MySQL复制为GTID复制

### MySQL需要配置的部分

1. 开启binlog并设置为row格式：

操作步骤：

log-bin=mysqlbin.log

binlog\_format=ROW

server-id=1

1. 需要在MySQL端开启GTID模式

MySQL版本在5.6以上

SET GLOBAL ENFORCE\_GTID\_CONSISTENCY = 'WARN';

SET GLOBAL ENFORCE\_GTID\_CONSISTENCY = 'ON';

SET GLOBAL GTID\_MODE = 'OFF\_PERMISSIVE';

SET GLOBAL GTID\_MODE = 'ON\_PERMISSIVE';

SET GLOBAL GTID\_MODE = 'ON';

### 建库语法

CREATE DATABASE ckdb ENGINE = MaterializeMySQL('[mysql-host]:[mysql-port]', '[mysql-database]', '[mysql-user]', '[mysql-password]');

# 表引擎

## MergeTree家族

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 系列的引擎被设计用于插入极大量的数据到一张表当中。数据可以以数据片段的形式一个接着一个的快速写入，数据片段在后台按照一定的规则进行合并。相比在插入时不断修改（重写）已存储的数据，这种策略会高效很多。

主要特点:

① 存储的数据按主键排序。这使得你能够创建一个小型的稀疏索引来加快数据检索。

②  支持数据分区，如果指定了 分区键 的话。在相同数据集和相同结果集的情况下 ClickHouse 中某些带分区的操作会比普通操作更快。查询中指定了分区键时 ClickHouse 会自动截取分区数据。这也有效增加了查询性能。

③ 支持数据副本。ReplicatedMergeTree 系列的表提供了数据副本功能。

④ 支持数据采样。

### MergeTree

#### 建表语法

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

(

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

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

...

INDEX index\_name1 expr1 TYPE type1(...) GRANULARITY value1,

INDEX index\_name2 expr2 TYPE type2(...) GRANULARITY value2

) ENGINE = MergeTree()

ORDER BY expr

[PARTITION BY expr]

[PRIMARY KEY expr]

[SAMPLE BY expr]

[TTL expr [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'], ...]

[SETTINGS name=value, ...]

#### 参数详解

① ENGINE - 引擎名和参数。 ENGINE = MergeTree(). MergeTree 引擎没有参数。

② ORDER BY — 排序键。可以是一组列的元组或任意的表达式。 例如: ORDER BY (CounterID, EventDate) 。如果没有使用 PRIMARY KEY 显式的指定主键，ClickHouse 会使用排序键作为主键。如果不需要排序，可以使用 ORDER BY tuple(). 参考 选择主键

③ PARTITION BY — 分区键 。要按月分区，可以使用表达式 toYYYYMM(date\_column) ，这里的 date\_column 是一个 Date 类型的列。分区名的格式会是 "YYYYMM" 。分区键可以单个列字段，也可以是通过元祖形式使用的多个列字段，还可以支持使用列表达式。若不声明分区键则clickhouse会生成一个名为all的分区。合理使用分区 可以有效减少查询数据文件的扫描范围。

④ PRIMARY KEY - 主键，如果要 选择与排序键不同的主键，可选。默认情况下主键跟排序键（由 ORDER BY 子句指定）相同。因此，大部分情况下不需要再专门指定一个 PRIMARY KEY 子句。

⑤ SAMPLE BY — 用于抽样的表达式。如果要用抽样表达式，主键中必须包含这个表达式。例如：SAMPLE BY intHash32(UserID) ORDER BY (CounterID, EventDate, intHash32(UserID)) 。

⑥ TTL（Time To Live） 指定行存储的持续时间并定义数据片段在硬盘和卷上的移动逻辑的规则列表，可选。表达式中必须存在至少一个 Date 或 DateTime 类型的列，比如：TTL date + INTERVAl 1 DAY 规则的类型 DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'指定了当满足条件（到达指定时间）时所要执行的动作：移除过期的行，还是将数据片段（如果数据片段中的所有行都满足表达式的话）移动到指定的磁盘（TO DISK 'xxx') 或 卷（TO VOLUME 'xxx'）。默认的规则是移除（DELETE）。可以在列表中指定多个规则，但最多只能有一个DELETE的规则。更多细节，请查看 表和列的 TTL

⑦ SETTINGS — 控制 MergeTree 行为的额外参数：

index\_granularity — 索引粒度。索引中相邻的『标记』间的数据行数。默认值，8192 。参考数据存储。

index\_granularity\_bytes — 索引粒度，以字节为单位，默认值: 10Mb。如果想要仅按数据行数限制索引粒度, 请设置为0(不建议)。

enable\_mixed\_granularity\_parts — 是否启用通过 index\_granularity\_bytes 控制索引粒度的大小。在19.11版本之前, 只有 index\_granularity 配置能够用于限制索引粒度的大小。当从具有很大的行（几十上百兆字节）的表中查询数据时候，index\_granularity\_bytes 配置能够提升ClickHouse的性能。如果你的表里有很大的行，可以开启这项配置来提升SELECT 查询的性能。

use\_minimalistic\_part\_header\_in\_zookeeper — 是否在 ZooKeeper 中启用最小的数据片段头 。如果设置了 use\_minimalistic\_part\_header\_in\_zookeeper=1 ，ZooKeeper 会存储更少的数据。更多信息参考『服务配置参数』这章中的 设置描述 。

min\_merge\_bytes\_to\_use\_direct\_io — 使用直接 I/O 来操作磁盘的合并操作时要求的最小数据量。合并数据片段时，ClickHouse 会计算要被合并的所有数据的总存储空间。如果大小超过了 min\_merge\_bytes\_to\_use\_direct\_io 设置的字节数，则 ClickHouse 将使用直接 I/O 接口（O\_DIRECT 选项）对磁盘读写。如果设置 min\_merge\_bytes\_to\_use\_direct\_io = 0 ，则会禁用直接 I/O。默认值：10 \* 1024 \* 1024 \* 1024 字节。

merge\_with\_ttl\_timeout — TTL合并频率的最小间隔时间，单位：秒。默认值: 86400 (1 天)。

write\_final\_mark — 是否启用在数据片段尾部写入最终索引标记。默认值: 1（不建议更改）。

merge\_max\_block\_size — 在块中进行合并操作时的最大行数限制。默认值：8192

storage\_policy — 存储策略。 参见 使用具有多个块的设备进行数据存储.

min\_bytes\_for\_wide\_part,min\_rows\_for\_wide\_part 在数据片段中可以使用Wide格式进行存储的最小字节数/行数。你可以不设置、只设置一个，或全都设置。参考：数据存储

### ReplacingMergeTree

该引擎和 MergeTree 的不同之处在于它会删除排序键值相同的重复项。

数据的去重只会在数据合并期间进行。合并会在后台一个不确定的时间进行，因此你无法预先作出计划。有一些数据可能仍未被处理。尽管你可以调用 OPTIMIZE 语句发起计划外的合并，但请不要依靠它，因为 OPTIMIZE 语句会引发对数据的大量读写。

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。

#### 建表语法

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

(

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

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

...

) ENGINE = ReplacingMergeTree([ver])

[PARTITION BY expr]

[ORDER BY expr]

[SAMPLE BY expr]

[SETTINGS name=value, ...]

#### 参数详解

其他语法和MergeTree引擎一直这里不再赘述，只介绍参数ver

ver — 版本列。类型为 UInt\*, Date 或 DateTime。可选参数。在数据合并的时候，ReplacingMergeTree 从所有具有相同排序键的行中选择一行留下：

如果 ver 列未指定，保留最后一条。

如果 ver 列已指定，保留 ver 值最大的版本。

### SummingMergeTree

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，对于不可加的列则会在现有的值中任选一个。

#### 建表语法

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

(

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

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

...

) ENGINE = SummingMergeTree([columns])

[PARTITION BY expr]

[ORDER BY expr]

[SAMPLE BY expr]

[SETTINGS name=value, ...]

#### 参数详解

columns - 包含了将要被汇总的列的列名的元组。可选参数。  
所选的列必须是数值类型，并且不可位于主键中。

如果没有指定 `columns`，ClickHouse 会把所有不在主键中的数值类型的列都进行汇总。

如果用于汇总的所有列中的值均为0，则该行会被删除。

如果列不在主键中且无法被汇总，则会在现有的值中任选一个。

主键所在的列中的值不会被汇总。

### AggregatingMergeTree

#### 介绍

 该引擎继承自 MergeTree，并改变了数据片段的合并逻辑。 ClickHouse 会将相同主键的所有行（在一个数据片段内）替换为单个存储一系列聚合函数状态的行。可以使用 AggregatingMergeTree 表来做增量数据统计聚合，包括物化视图的数据聚合。引擎需使用 AggregateFunction 类型来处理所有列。如果要按一组规则来合并减少行数，则使用 AggregatingMergeTree 是合适的。对于AggregatingMergeTree不能直接使用insert来查询写入数据。一般是用insert select。但更常用的是创建物化视图

#### 建表语法

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

(

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

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

...

) ENGINE = AggregatingMergeTree()

[PARTITION BY expr]

[ORDER BY expr]

[SAMPLE BY expr]

[TTL expr]

[SETTINGS name=value, ...]

#### 插入数据

要插入数据，需使用带有 -State- 聚合函数的 [INSERT SELECT](https://clickhouse.tech/docs/zh/sql-reference/statements/insert-into/) 语句

直接insert into数据会报错：类似于 Cannot convert Float64 to AggregateFunction(sum, Float64)

#### 查询数据

直接查询只能看到主键和聚合函数AggregateFunction()的别名和乱码

正确的查询方式是

从 AggregatingMergeTree 表中查询数据时，需使用 GROUP BY 子句并且要使用与插入时相同的聚合函数，但后缀要改为 -Merge 。

#### 用法示例

##### 建表

CREATE TABLE tb\_test\_AggregatingMergeTree\_table

(

`brandId` Int32,

`shopId` Int32,

`saleMoney` AggregateFunction(sum, Float32),

`saleQty` AggregateFunction(sum, Int32),

`saleNum` AggregateFunction(count, UInt8),

`vipNum` AggregateFunction(uniq, UInt64)

)

ENGINE = AggregatingMergeTree()

PARTITION BY (brandId, shopId)

ORDER BY (brandId, shopId)

##### 插入数据

INSERT INTO tb\_test\_AggregatingMergeTree\_table SELECT

brandId,

shopId,

sumState(saleMoney) AS saleMoney,

sumState(saleQty) AS saleQty,

countState(1) AS saleNum,

uniqState(vipId) AS vipNum

FROM tb\_test\_MergeTree\_basic

GROUP BY

brandId,

shopId;

##### 查询数据

SELECT

brandId,

shopId,

sumMerge(saleMoney) AS saleMoney,

sumMerge(saleQty) AS saleQty,

countMerge(saleNum) AS saleNum,

uniqMerge(vipNum) AS vipNum

FROM tb\_test\_AggregatingMergeTree\_table

GROUP BY

brandId,

shopId;

#### 物化视图

##### 物化视图创建示例

创建一个跟踪tb\_test\_MergeTree\_basic表（MergeTree引擎）的物化视图

CREATE MATERIALIZED VIEW tb\_test\_AggregatingMergeTree\_view

ENGINE = AggregatingMergeTree()

PARTITION BY (brandId, shopId)

ORDER BY (brandId, shopId) AS

SELECT

brandId,

shopId,

sumState(saleMoney) AS saleMoney,

sumState(saleQty) AS saleQty,

countState(1) AS saleNum,

uniqState(vipId) AS vipNum

FROM tb\_test\_MergeTree\_basic

GROUP BY

brandId,

shopId

##### 注意点

1. 创建视图前已经存在的数据不能跟踪
2. 可以自动跟踪聚合创建视图后再插入的数据

### CollapsingMergeTree

该引擎继承于 MergeTree，并在数据块合并算法中添加了折叠行的逻辑。CollapsingMergeTree 会异步的删除（折叠）这些除了特定列 Sign 有 1 和 -1 的值以外，其余所有字段的值都相等的成对的行。没有成对的行会被保留。也就是说CollapsingMergeTree引擎有个状态列sign，这个值1为”状态”行，-1为”取消”行，对于数据只关心状态列为状态的数据，不关心状态列为取消的数据

#### 建表语法

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

(

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

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

...

) ENGINE = CollapsingMergeTree(sign)

[PARTITION BY expr]

[ORDER BY expr]

[SAMPLE BY expr]

[SETTINGS name=value, ...]

#### 参数详解

sign — 类型列的名称： 1 是«状态»行，-1 是«取消»行。

列数据类型 — Int8。其他数据类型会报错，比如 Int32报错：Code: 169. DB::Exception: Received from xx.xx.x.x:9000. DB::Exception: Sign column (sign) for storage CollapsingMergeTree must have type Int8. Provided column of type Int32..

参数必填项。如果不填建表报错：Code: 42. DB::Exception: Received from xx.xx.x.x:9000. DB::Exception: With extended storage definition syntax storage CollapsingMergeTree requires 1 parameters:

### VersionedCollapsingMergeTree

#### 介绍

这个引擎和CollapsingMergeTree差不多，只是对CollapsingMergeTree引擎加了一个版本，比如可以适用于非实时用户在线统计，统计每个节点用户在在线业务

这个引擎允许快速写入不断变化的对象状态。

删除后台中的旧对象状态。 这显着降低了存储体积。

#### 建表语法

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

(

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

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

...

) ENGINE = VersionedCollapsingMergeTree(sign, version)

[PARTITION BY expr]

[ORDER BY expr]

[SAMPLE BY expr]

[SETTINGS name=value, ...]

#### 参数介绍

sign — 指定行类型的列名: 1 是一个 “state” 行, -1 是一个 “cancel” 行

列数据类型应为 Int8.

version — 指定对象状态版本的列名。

列数据类型应为 UInt\*.

### 副本表

只有 MergeTree 系列里的表可支持副本：

* ReplicatedMergeTree
* ReplicatedSummingMergeTree
* ReplicatedReplacingMergeTree
* ReplicatedAggregatingMergeTree
* ReplicatedCollapsingMergeTree
* ReplicatedVersionedCollapsingMergetree
* ReplicatedGraphiteMergeTree

#### 建表语法

CREATE TABLE table\_name

(

EventDate DateTime,

CounterID UInt32,

UserID UInt32

) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/table\_name', '{replica}')

PARTITION BY toYYYYMM(EventDate)

ORDER BY (CounterID, EventDate, intHash32(UserID))

SAMPLE BY intHash32(UserID)

#### 参数说明

这些参数可以包含宏替换的占位符，即大括号的部分。它们会被替换为配置文件里 ‘macros’

那部分配置的值。示例：

<macros>

<layer>05</layer>

<shard>02</shard>

<replica>example05-02-1.yandex.ru</replica>

</macros>

ZooKeeper 中该表的路径对每个可复制表都要是唯一的。不同分片上的表要有不同的路径。  
这种情况下，路径包含下面这些部分：

/clickhouse/tables/ 是公共前缀，我们推荐使用这个。

{layer}-{shard} 是分片标识部分。在此示例中，由于 Yandex.Metrica 集群使用了两级分片，所以它是由两部分组成的。但对于大多数情况来说，你只需保留 {shard} 占位符即可，它会替换展开为分片标识。

table\_name 是该表在 ZooKeeper 中的名称。使其与 ClickHouse 中的表名相同比较好。 这里它被明确定义，跟 ClickHouse 表名不一样，它并不会被 RENAME 语句修改。  
HINT：你可以在前面添加一个数据库名称 table\_name 也是 例如。 db\_name.table\_name

副本名称用于标识同一个表分片的不同副本。你可以使用服务器名称，如上例所示。同个分片中不同副本的副本名称要唯一。

你也可以显式指定这些参数，而不是使用宏替换。对于测试和配置小型集群这可能会很方便。但是，这种情况下，则不能使用分布式 DDL 语句（ON CLUSTER）。

使用大型集群时，我们建议使用宏替换，因为它可以降低出错的可能性。

## 日志引擎

Log

TinyLog

StripeLog

### 相同点

1.数据存储在磁盘上。

2.写入时将数据追加在文件末尾。

3.不支持[突变](https://clickhouse.tech/docs/zh/engines/table-engines/log-family/" \l "alter-mutations)操作。

4.不支持索引。这意味着 `SELECT` 在范围查询时效率不高。

5.非原子地写入数据。如果某些事情破坏了写操作，例如服务器的异常关闭，你将会得到一张包含了损坏数据的表。

### 不同点

Log 和 StripeLog 引擎支持：

1.并发访问数据的锁。

`INSERT` 请求执行过程中表会被锁定，并且其他的读写数据的请求都会等待直到锁定被解除。如果没有写数据的请求，任意数量的读请求都可以并发执行。

2.并行读取数据。

在读取数据时，ClickHouse 使用多线程。 每个线程处理不同的数据块。

Log 引擎为表中的每一列使用不同的文件。StripeLog 将所有的数据存储在一个文件中。因此 StripeLog 引擎在操作系统中使用更少的描述符，但是 Log 引擎提供更高的读性能。

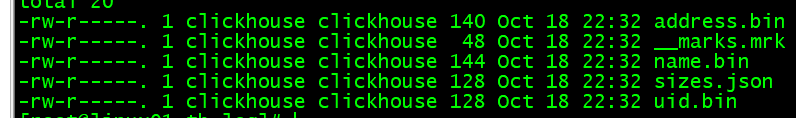
TinyLog 引擎是该系列中最简单的引擎并且提供了最少的功能和最低的性能。TinyLog 引擎不支持并行读取和并发数据访问，并将每一列存储在不同的文件中。它比其余两种支持并行读取的引擎的读取速度更慢，并且使用了和 Log 引擎同样多的描述符。你可以在简单的低负载的情景下使用它。

### Log

#### 介绍

日志与 TinyLog 的不同之处在于，«标记» 的小文件与列文件存在一起。这些标记写在每个数据块上，并且包含偏移量，这些偏移量指示从哪里开始读取文件以便跳过指定的行数。这使得可以在多个线程中读取表数据。对于并发数据访问，可以同时执行读取操作，而写入操作则阻塞读取和其它写入。Log 引擎不支持索引。同样，如果写入表失败，则该表将被破坏，并且从该表读取将返回错误。Log 引擎适用于临时数据，write-once 表以及测试或演示目的。

#### 数据目录



### StripeLog

#### 介绍

在你需要写入许多小数据量（小于一百万行）的表的场景下使用这个引擎

#### 写数据

StripeLog 引擎将所有列存储在一个文件中。对每一次 Insert 请求，ClickHouse 将数据块追加在表文件的末尾，逐列写入。

ClickHouse 为每张表写入以下文件：

data.bin：数据文件。

index.mrk ：带标记的文件。标记包含了已插入的每个数据块中每列的偏移量。

StripeLog 引擎不支持 ALTER UPDATE 和 ALTER DELETE 操作。

#### 读数据

带标记的文件使得 ClickHouse 可以并行的读取数据。这意味着 SELECT 请求返回行的顺序是不可预测的。我们可以使用 ORDER BY 子句对结果行进行排序。

每次 INSERT 请求都会在 data.bin 文件中创建一个数据块。

ClickHouse 在查询数据时使用多线程。每个线程读取单独的数据块并在完成后独立的返回结果行。这样的结果是，大多数情况下，输出中块的顺序和输入时相应块的顺序是不同的。

### TinyLog

#### 介绍

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。

并发数据访问不受任何限制：  
- 如果同时从表中读取并在不同的查询中写入，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。查询在单个流中执行。换句话说，此引擎适用于相对较小的表（建议最多1,000,000行）。如果您有许多小表，则使用此表引擎是适合的，因为它比Log引擎更简单（需要打开的文件更少）。当您拥有大量小表时，可能会导致性能低下，但在可能已经在其它 DBMS 时使用过，则您可能会发现切换使用 TinyLog 类型的表更容易。不支持索引。

在 Yandex.Metrica 中，TinyLog 表用于小批量处理的中间数据。

## 集成引擎

### JDBC

#### 介绍

允许CH通过 [JDBC](https://en.wikipedia.org/wiki/Java_Database_Connectivity" \t "https://clickhouse.tech/docs/zh/engines/table-engines/integrations/jdbc/_blank) 连接到外部数据库。

要实现JDBC连接，CH需要使用以后台进程运行的程序 [clickhouse-jdbc-bridge](https://github.com/alex-krash/clickhouse-jdbc-bridge" \t "https://clickhouse.tech/docs/zh/engines/table-engines/integrations/jdbc/_blank)。

该引擎支持 [Nullable](https://clickhouse.tech/docs/zh/sql-reference/data-types/nullable/) 数据类型。

#### 建表语法

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

(

columns list...

)

ENGINE = JDBC(dbms\_uri, external\_database, external\_table)

#### 参数详解

dbms\_uri ：外部DBMS的uri.

格式: jdbc:<driver\_name>://<host\_name>:<port>/?user=<username>&password=<password>.  
MySQL示例: jdbc:mysql://localhost:3306/?user=root&password=root.

external\_database ：外部DBMS的数据库名.

external\_table： external\_database中的外部表名.

### ODBC

#### linux如何通过ODBC连接数据库

1.下载UnixODBC安装包

yum install -y unixODBC.x86\_64

2.下载要连接数据库的 Connector/ODBC安装包

yum install -y postgresql-odbc.x86\_64

3.配置/etc/odbc.ini文件

vim /etc/odbc.ini

[pg]

Description = Test to pg

Driver = PostgreSQL #与/etc/odbcinst.ini 中一致

Database = postgres

Servername = 192.168.9.139

UserName = admin

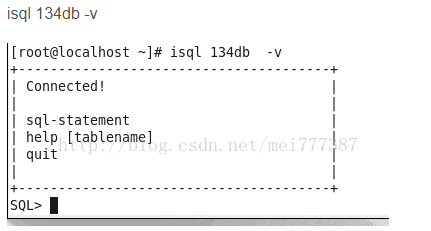
Password = admin

Port = 5432

ReadOnly = 0

4.测试是否成功

isql pg -v



#### 介绍

允许ClickHouse通过[ODBC](https://en.wikipedia.org/wiki/Open_Database_Connectivity" \t "https://clickhouse.tech/docs/zh/engines/table-engines/integrations/odbc/_blank)方式连接到外部数据库.

为了安全地实现ODBC连接，ClickHouse使用了一个独立程序 clickhouse-odbc-bridge.

该引擎支持 [可为空](https://clickhouse.tech/docs/zh/sql-reference/data-types/nullable/) 的数据类型。

#### 建表语法

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

name1 [type1],

name2 [type2],

...)ENGINE = ODBC(connection\_settings, external\_database, external\_table)

表结构可以与源表结构不同:

列名应与源表中的列名相同，但您可以按任何顺序使用其中的一些列。

列类型可能与源表中的列类型不同。 ClickHouse尝试将数值[映射](https://clickhouse.tech/docs/zh/sql-reference/functions/type-conversion-functions/" \l "type_conversion_function-cast) 到ClickHouse的数据类型。

#### 参数详解

connection\_settings：odbc.ini中具有连接设置的节的名称

external\_database：数据库

external\_table：数据表

#### 用法示例

通过ODBC从本地安装的MySQL中检索数据

本示例针对Ubuntu Linux18.04和MySQL服务器5.7进行检查。

请确保安装了unixODBC和MySQL连接器。

1.默认情况下（如果从软件包安装），ClickHouse以用户clickhouse的身份启动 . 因此，您需要在MySQL服务器中创建和配置此用户。

$ sudo mysql

mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse';

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION;

2.然后在/etc/odbc.ini中配置连接 .

$ cat /etc/odbc.ini

[mysqlconn]

DRIVER = /usr/local/lib/libmyodbc5w.so

SERVER = 127.0.0.1

PORT = 3306

DATABASE = test

USERNAME = clickhouse

PASSWORD = clickhouse

3.从安装的unixodbc中使用 isql 实用程序来检查连接情况。

$ isql -v mysqlconn

+-------------------------+| Connected! || |

...

4.在clickhouse中建立ODBC引擎表，访问MySQL表中的数据。

CREATE TABLE odbc\_t(

`int\_id` Int32,

`float\_nullable` Nullable(Float32))

ENGINE = ODBC('DSN=mysqlconn', 'test', 'test')

### HDFS

#### 建表语法

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

name1 [type1],

name2 [type2],

...)ENGINE = HDFS(URI,format)

#### 参数详解

URI：hdfs中的文件路径

URI示例：'hdfs://node01:8020/user/hive/warehouse/test.db/hdfsTest/a.csv'

format：文件格式

目前支持的文件格式:CSV,TSV,JSON.....

#### url的路径匹配模式

\*:匹配任意字符 /user/hdfsTest/a\*读取所有a开头的文件

?:匹配单个字符 /user/hdfsTest/a?.txt 读取比如aa.txt,ab.txt…

{a,b,c}:匹配其中任一 /user/hdfsTest/{a,b,c}.txt 匹配a.txt b.txt和c.txt

{x…x}: 匹配数字区间 /user/hdfsTest/{1…5}.txt 匹配1.txt 2.txt … 5.txt

#### 用法示例

create table hdfsTest(id Int16,name String)

engine=HDFS('hdfs://node01:8020/user/hive/warehouse/test.db/hdfsTest/a.csv','CSV');

insert into hdfsTest values(1,'zhangsan');

总结：url指向的路径中是否包含有内容,如果没有内容,那么可读可写,如果有内容,则只读.

#### 异常

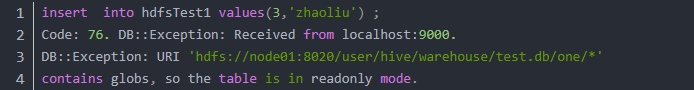
1. insert into 报不能打开,权限异常的话,更改文件夹的权限:

hdfs dfs -chmod 777 /user/hive/warehouse/test.db/hdfsTest/

或者

hdfs dfs -chown -R clickhouse:clickhouse /user/hive/warehouse/test.db/hdfsTest/

1. url指向的路径有内容,只能读不能写，报异常如图片所示



#### HDFS表函数

读取hdfs数据还有一种表函数:hdfs(url,format,structure)

create table hdfsTest1(id Int16,name String)

engine = HDFS('hdfs://node01:8020/user/hive/warehouse/test.db/one/\*','CSV');

可以使用下面的SQL替代

create table hdfsTest3 engine = MergeTree order by id

as select \* from

hdfs('hdfs://node01:8020/user/hive/warehouse/test.db/one/\*','CSV','id Int8,name String') ;

### Kafka

#### 建表语句

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

name1 [type1],

name2 [type2],

...)ENGINE = Kafka SETTINGS

kafka\_broker\_list = 'localhost:9092',

kafka\_topic\_list = 'topic1,topic2',

kafka\_group\_name = 'group1',

kafka\_format = 'JSONEachRow',

kafka\_row\_delimiter = '\n',

kafka\_schema = '',

kafka\_num\_consumers = 2

#### 参数详解

##### 必要参数

kafka\_broker\_list – 以逗号分隔的 brokers 列表 (localhost:9092)。

kafka\_topic\_list – topic 列表 (my\_topic)。

kafka\_group\_name – Kafka 消费组名称 (group1)。如果不希望消息在集群中重复，请在每个分片中使用相同的组名。

kafka\_format – 消息体格式。使用与 SQL 部分的 FORMAT 函数相同表示方法，例如 JSONEachRow。了解详细信息，请参考 Formats 部分。

##### 可选参数

kafka\_row\_delimiter - 每个消息体（记录）之间的分隔符。

kafka\_schema – 如果解析格式需要一个 schema 时，此参数必填。例如，[普罗托船长](https://capnproto.org/" \t "https://clickhouse.tech/docs/zh/engines/table-engines/integrations/kafka/_blank) 需要 schema 文件路径以及根对象 schema.capnp:Message 的名字。

kafka\_num\_consumers – 单个表的消费者数量。默认值是：1，如果一个消费者的吞吐量不足，则指定更多的消费者。消费者的总数不应该超过 topic 中分区的数量，因为每个分区只能分配一个消费者。

### MySQL

MySQL 引擎可以对存储在远程 MySQL 服务器上的数据执行 SELECT 查询。

#### 建表语法

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

(

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

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

...

) ENGINE = MySQL('host:port', 'database', 'table', 'user', 'password'[, replace\_query, 'on\_duplicate\_clause']);

#### 参数详解

host:port MySQL服务器的地址和端口

database MySQL中的数据库名

table 表示需要映射的MySQL表名称

user 用户名

password 用户名对应的密码

replace\_query 默认为0，对应于MySQL的REPLACE INTO 语法；若为1则会用replace into 代替insert into

on\_duplicate\_clause 默认为0 对应MySQL的 ON DUPLICATE KEY语法，若需要使用该设置则必须将replace\_query 设置为0

## 特别

### 分布式表引擎

#### 介绍

在clickhouse分布式集群中，我们通常需要先创建本地表再创建分布式表。因为本地表储存数据，而分布式表只是作为一个查询引擎本身不存储任何数据，查询时将sql发送到所有集群分片，然后进行进行处理和聚合后将结果返回给客户端。数据插入时能够⾃动的路由数据⾄集群中的各个节点，即分布式表需要和其他数据表⼀起协同⼯作。分布式表会将接收到的读写任务，分发到各个本地表，而实际上数据的存储也是保存在各个节点的本地表中。

远程服务器不仅用于读取数据，还会对尽可能数据做部分处理。  
例如，对于使用 GROUP BY 的查询，数据首先在远程服务器聚合，之后返回聚合函数的中间状态给查询请求的服务器。再在请求的服务器上进一步汇总数据。

#### 引擎参数详解

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

cluster：集群名称，在对分布式表执⾏读写的过程中，它会使⽤集群的配置信息来找到相应的host节点。

database，table：数据库和本地表名称，用于将分布式表映射到本地表上。

sharding\_key: 分⽚键，分布式表会按照这个规则，将数据分发到各个本地表中。

### Merge引擎

Merge引擎与MergeTree引擎一点儿关系没有,它类似于union all的功能,将不同引擎表中的数据union到一起.

#### 引擎参数

engine = Merge(参数1，参数2)

参数1：指明库名  
参数2 : 指明表名,支持正则表示

#### 注意点

1. 被merge的表字段名不一样将会报错,但是字段名一样而类型不一样,并且类型可以自动转化的情况下,不会报错.
2. Merge引擎有一列隐藏字段 \_table,记录的是数据来源的表

所以merge引擎可以根据\_table来过滤,如:

select \* from merge表 where \_table ='mergeTest1';

### 随机数生成表引擎

ENGINE = GenerateRandom(random\_seed, max\_string\_length, max\_array\_length)

max\_array\_length：array列的最大长度

max\_string\_length：string数据的最大长度

该引擎仅支持 SELECT 查询语句

该引擎支持能在表中存储的所有数据类型 [DataTypes](https://clickhouse.tech/docs/zh/sql-reference/data-types/) ，除了 LowCardinality 和 AggregateFunction.

### JOIN引擎

#### 建表语法

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

(

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

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

) ENGINE = Join(join\_strictness, join\_type, k1[, k2, ...])

#### 参数详解

join\_strictness – [JOIN 限制](https://clickhouse.tech/docs/zh/sql-reference/statements/select/join/" \l "select-join-types)，Any代表没有限制

join\_type – [JOIN 类型](https://clickhouse.tech/docs/zh/sql-reference/statements/select/join/" \l "select-join-types)：right，left，full

k1[, k2, ...] – 进行JOIN 操作时 USING语句用到的key列

### Memory表引擎

Memory 引擎以未压缩的形式将数据存储在 RAM 中。重新启动服务器时，表中的数据消失，表将变为空。

并发数据访问是同步的。锁范围小：读写操作不会相互阻塞。

不支持索引（已经很快了，不需要索引）。

查询是并行化的。

在简单查询上达到最大速率（超过10 GB /秒），因为没有磁盘读取，不需要解压缩或反序列化数据。（值得注意的是，在许多情况下，与 MergeTree 引擎的性能几乎一样高）。

通常，使用此表引擎是不合理的，但是，它可用于测试，以及在相对较少的行（最多约100,000,000）上需要最高性能的查询。

### File引擎

#### 介绍

数据源是以 Clickhouse 支持的一种输入格式（TabSeparated，Native等）存储数据的文件。

但是创建表语句并没有指定路径,所以没有办法指定路径进行数据访问.

数据存储在对应的数据库表下:  
/var/lib/clickhouse/data/{库名}/{表名}/data.{fomat}

#### 建表语句

create table tableName (col1 type,col2 type...)engine =File(format);

format格式常见的有：CSV,TSV,JSONEachRow

# 函数

## 算数函数

加：plus(a+b)

还可以将Date或DateTime与整数进行相加。在Date的情况下，和整数相加整数意味着添加相应的天数。对于DateTime，这意味着添加相应的秒数。

减: minus(a+b)

乘：multiply(a,b)

除: divide(a,b)

结果类型始终是浮点类型。  
它不是整数除法。对于整数除法，请使用’intDiv’函数。  
当除以零时，你得到’inf’，‘- inf’或’nan’。

余数：modulo(a, b)，a%b

计算除法后的余数。  
如果参数是浮点数，则通过删除小数部分将它们预转换为整数。  
其余部分与C++中的含义相同。截断除法用于负数。  
除以零或将最小负数除以-1时抛出异常。

整数除法：intDiv(a,b)

计算数值的商，向下舍入取整（按绝对值）。  
除以零或将最小负数除以-1时抛出异常。

intDivOrZero(a,b)

与’intDiv’的不同之处在于它在除以零或将最小负数除以-1时返回零。

变负数：negate(a), -a operator

通过改变数值的符号位对数值取反，结果总是有符号的

绝对值：abs(a)

计算数值（a）的绝对值。也就是说，如果a \< 0，它返回-a。对于无符号类型，它不执行任何操作。对于有符号整数类型，它返回无符号数。

gcd(a,b)

返回数值的最大公约数。  
除以零或将最小负数除以-1时抛出异常。

lcm(a,b)

返回数值的最小公倍数。  
除以零或将最小负数除以-1时抛出异常。

## 比较函数

比较函数始终返回0或1（UInt8）。

可以比较以下类型：

* 数字
* String 和 FixedString
* 日期
* 日期时间

以上每个组内的类型均可互相比较，但是对于不同组的类型间不能够进行比较。

## 逻辑函数

逻辑函数可以接受任何数字类型的参数，并返回UInt8类型的0或1。

当向函数传递零时，函数将判定为«false»，否则，任何其他非零的值都将被判定为«true»。

和，AND 运算符

或，OR 运算符

非，NOT 运算符[¶](https://clickhouse.tech/docs/zh/sql-reference/functions/logical-functions/" \l "not-not-operator" \o "Permanent link)

异或，XOR 运算符

## 类型转换函数

toInt(8|16|32|64)

toInt(8|16|32|64)OrZero

这个函数需要一个字符类型的入参，然后尝试把它转为Int (8 | 16 | 32 | 64)，如果转换失败直接返回0。

toInt(8|16|32|64)OrNull

如果转换失败直接返回NULL。

toUInt(8|16|32|64)OrZero

toUInt(8|16|32|64)OrNull

toFloat(32|64)

toFloat(32|64)OrZero

toFloat(32|64)OrNull[¶](https://clickhouse.tech/docs/zh/sql-reference/functions/type-conversion-functions/" \l "tofloat3264ornull" \o "Permanent link)

toInt(8|16|32|64)OrNull

toUInt(8|16|32|64)

toDate

toDateOrZero

toDateOrNull

toDateTime

toDateTimeOrZero

toDateTimeOrNull

toDecimal(32|64|128)

toDecimal(32|64|128)OrZero

toString()

查看数据类型：toTypeName()

## GEO函数

### 地球两点距离

使用[great-circle distance公式](https://en.wikipedia.org/wiki/Great-circle_distance" \t "https://clickhouse.tech/docs/zh/sql-reference/functions/geo/_blank)计算地球表面两点之间的距离。

greatCircleDistance(lon1Deg, lat1Deg, lon2Deg, lat2Deg)

输入参数

lon1Deg — 第一个点的经度，单位：度，范围： [-180°, 180°]。

lat1Deg — 第一个点的纬度，单位：度，范围： [-90°, 90°]。

lon2Deg — 第二个点的经度，单位：度，范围： [-180°, 180°]。

lat2Deg — 第二个点的纬度，单位：度，范围： [-90°, 90°]。

正值对应北纬和东经，负值对应南纬和西经。

返回值

地球表面的两点之间的距离，以米为单位。

当输入参数值超出规定的范围时将抛出异常。

### pointInPolygon

检查指定的点是否包含在指定的多边形中。

pointInPolygon((x, y), [(a, b), (c, d) ...], ...)

输入参数

(x, y) — 平面上某个点的坐标。[元组](https://clickhouse.tech/docs/zh/sql-reference/functions/geo/)类型，包含坐标的两个数字。

[(a, b), (c, d) ...] — 多边形的顶点。[阵列](https://clickhouse.tech/docs/zh/sql-reference/functions/geo/)类型。每个顶点由一对坐标(a, b)表示。顶点可以按顺时针或逆时针指定。顶点的个数应该大于等于3。同时只能是常量的。

该函数还支持镂空的多边形（切除部分）。如果需要，可以使用函数的其他参数定义需要切除部分的多边形。(The function does not support non-simply-connected polygons.)

返回值

如果坐标点存在在多边形范围内，则返回1。否则返回0。  
如果坐标位于多边形的边界上，则该函数可能返回1，或可能返回0。

### geohashEncode

将经度和纬度编码为geohash-string

geohashEncode(longitude, latitude, [**precision**])

输入参数

longitude - 要编码的坐标的经度部分。其值应在[-180°，180°]范围内

latitude - 要编码的坐标的纬度部分。其值应在[-90°，90°]范围内

precision - 可选，生成的geohash-string的长度，默认为12。取值范围为[1,12]。任何小于1或大于12的值都会默认转换为12。

返回值

坐标编码的字符串（使用base32编码的修改版本）。

### geohashDecode

将任何geohash编码的字符串解码为经度和纬度。

geohashDecode(encoded string)

* 输入值：encoded string - geohash编码的字符串。
* 输出值：(longitude, latitude) - 经度和纬度的Float64值的2元组。

## Nullable处理函数

isNull

isNotNull

ifNull

nullIf(x,y):

如果参数相等，则为NULL。

如果参数不相等，则为x值。

assumeNotNu(x)

将[可为空](https://clickhouse.tech/docs/zh/sql-reference/functions/functions-for-nulls/)类型的值转换为非Nullable类型的值。

参数：x---原始值

如果x不为NULL，返回非Nullable类型的原始值。

如果x为NULL，返回对应非Nullable类型的默认值。

toNullable(x)

将参数的类型转换为Nullable。

## 舍入函数

floor(x[,N]):向下取数

示例

floor(123.45, 1) = 123.4

floor(123.45, -1) = 120.

ceil(x[,N]):向上取数

round(x[, N]):四舍五入

## 字典函数

dictGetT('dict\_name', 'attr\_name', id)

使用’id’键获取dict\_name字典中attr\_name属性的值。dict\_name和attr\_name是常量字符串。id必须是UInt64。

如果字典中没有id键，则返回字典描述中指定的默认值。

dictGetTOrDefault('dict\_name', 'attr\_name', id, default)

与dictGetT函数相同，但默认值取自函数的最后一个参数。

dictHas('dict\_name', id)

检查字典是否存在指定的id。如果不存在，则返回0;如果存在，则返回1。

## 字符串函数

empty():判断字符串是空为1，否则为0

notEmpty

length

lower

upper

reverse

substring(s, offset, length)

appendTrailingCharIfAbsent(s, c)

## 字符串拆分合并函数

splitByChar(separator, s):以单个字符分割字符串

splitByString(separator, s):以单个或多个字符分割字符串

concat(s1,s2,…)

## 字符串搜索函数

match(haystack,pattern):字符串正则匹配，返回0或1

extract(haystack,pattern):返回匹配到的第一个子串

extractAll(haystack,pattern):返回匹配到的所有子串，输出列表

like(haystack,pattern):匹配到的数据返回1，否则返回0

notLike(haystack, pattern):与like()函数相反

## 字符串替换函数

replaceOne(haystack,pattern,replacement):替换第一个匹配到的pattern

replaceAll(haystack,pattern,replacement):替换所有匹配到的pattern

replaceRegexpOne(haystack, pattern, replacement):正则匹配替换第一个匹配到的pattern

replaceRegexpAll(haystack,pattern,replacement):正则匹配替换所有匹配到的pattern

## 数学函数

e()

pi()

log(x):返回log以e为底的对数值

log2(x)

sqrt(x):对x开平方

cbrt(x):对x开立方

pow(x, y):返回x的y次方

## 数组函数

empty:对于空数组返回1，对于非空数组返回0。  
结果类型是UInt8。  
该函数也适用于字符串。

notEmpty：对于空数组返回0，对于非空数组返回1。  
结果类型是UInt8。  
该函数也适用于字符串。

length

返回数组中的元素个数。  
结果类型是UInt64。  
该函数也适用于字符串。

range(start, end [, step]):生成整型数组  
以防万一，如果在数据块中创建总长度超过100,000,000个元素的数组，则抛出异常。

示例：range(9) 返回：[0,1,2,3,4,5,6,7,8]

range(2,9) 返回：[2,3,4,5,6,7,8]

range(2,17,3) 返回：[2,5,8,11,14]

array() 或者[]:数组创建

arrayElement(arr, n) 或者 arr[]：获取数据元素

has(arr, elem)：是否包含某一个元素

hasAll(set, subset)：是否包含某几个元素

hasAny(array1, array2)：判断两个数组的交集元素

hasSubstr(array1, array2)：判断两个数组顺序是否一样

indexOf(arr, x)：数组中第一个元素首次出现的位置

countEqual(arr, x)：数组中某元素出现的次数

arrayEnumerate(arr)：返回数组下标

示例：SELECT arrayEnumerate([1, 2, 3, 1, 1, 1]) 返回：[1,2,3,4,5,6]

arrayEnumerateUniq(arr, …)：统计元素出现的次数

示例：SELECT arrayEnumerate([1, 2, 3, 1, 1, 1]) 返回：[1,1,1,2,3,4]

arrayEnumerateDense(arr)：数组元素出现的次数和位置

示例：SELECT arrayEnumerateDense( [1, 5, 23, 5, 5] ) ; 返回：[1,2,3,2,2]

arrayPopBack：删除数组中最后一个元素

arrayPopFront：删除数组中第一个元素

arrayPushBack(array, single\_value)：在数组中的末尾加入元素

arrayPushFront(array, single\_value)：在数组中的前端加入元素

arrayResize(array, size[, extender])：修改数组的长度

arrayConcat(arrays)：数组的拼接

arraySlice(array, offset[, length])：数据的切片

arraySort([func,] arr, …)：数据的排序（正向排序）

arrayReverseSort([func,] arr, …)：数组的反向排序

arrayUniq(arr, …)：数据的元素去重

arrayJoin(arr)：数据的连接

arrayDifference(array)：数组元素的差值

arrayDistinct(array)：数组元素的去重

arrayIntersect(arr)：数组元素的交集

arrayReduce(agg\_func, arr1, arr2, arrN)：数组元素的reduce

arrayReduceInRanges：数组元素的reduce

示例： select arrayReduceInRanges('max',[(1,3),(2,9)],[5,6,7,8,9,10]);

返回：[7,10]

第一个参数：聚合函数，用’ ‘包裹

第二个参数：数组，数组每个元素是个元组

第三个参数：数组

arrayReverse(arr)：数组的反转

reverse(arr)：数组的反转

arrayFlatten：数组的展平（二维数组变一维数组）

arrayCompact：数组的紧凑（对数组内数据实现相邻去重）

示例：SELECT arrayCompact([1, 1, 2, 2, 1, 2, 3, 3]) 返回：[1,2,1,2,3]

arrayWithConstant：生成一个指定长度的数组

示例：SELECT arrayWithConstant( 3, 'a') 返回：['a','a','a']

arrayStringConcat：将数组元素按照给定分隔符进行拼接，返回拼接后的字符串（数组元素必须为String类型）

示例：SELECT arrayStringConcat( ['2020','12','19'], '-') 返回：2020-12-19

arraySlice：对数组进行切割 ，后面两个参数分别是切割的位置和切割后的长度

示例：SELECT arraySlice([1, 2, 3, 6, 34, 3, 11], -3, 2) 返回：[34,3]

SELECT arraySlice([1, 2, 3, 6, 34, 3, 11], 3, 2) 返回：[3,6]

arrayDifference：计算数组中前后两个值的差值部分，该位=当前-前者（不包括第一个数，结果第一位默认为0）

示例：SELECT arrayDifference( [1,2,3,6,34,3,11] )

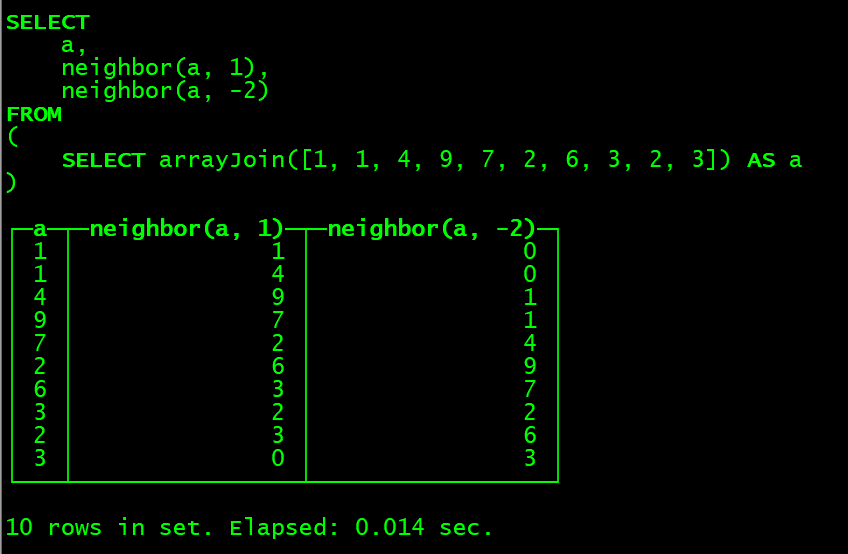
返回：[0,1,1,3,28,-31,8]

runningDifference:计算某一列前后数值的差值，该位=当前-前者（不包括第一个数，结果第一位默认为0）

neighbor():取其他行数据，类似于Hive窗口函数

示例：-- +n 表示取后面n行 -n 表示取前面第n行数据

SELECT a, neighbor( a,+1 ), neighbor( a,-2 ) from (SELECT arrayJoin( [1,1,4,9,7,2,6,3,2,3] ) as a);



## 高阶函数

arrayMap(func, arr1, …)

示例：SELECT arrayMap(y -> (y \* 10), [1, 2, 3, 4])

返回：[10,20,30,40]

arrayFilter(func, arr1, …)

示例：SELECT arrayFilter(x -> x LIKE '%World%', ['Hello', 'abc World']) AS res

返回： ['abc World']

arrayCount([func,] arr1, …)

返回数组arr中非零元素的数量，如果指定了’func’，则通过’func’的返回值确定元素是否为非零元素。

arrayExists([func,] arr1, …)

返回数组’arr’中是否存在非零元素，如果指定了’func’，则使用’func’的返回值确定元素是否为非零元素。

arrayAll([func,] arr1, …)

返回数组’arr’中是否存在为零的元素，如果指定了’func’，则使用’func’的返回值确定元素是否为零元素。

如果存在为零的元素，则返回0

如果不存在为零的元素，则返回1

arraySum([func,] arr1, …)

计算arr数组的总和，如果指定了’func’，则通过’func’的返回值计算数组的总和。

arrayFirst(func, arr1, …)

返回数组中第一个匹配的元素，函数使用’func’匹配所有元素，直到找到第一个匹配的元素。

arrayFirstIndex(func, arr1, …)

返回数组中第一个匹配的元素的下标索引，函数使用’func’匹配所有元素，直到找到第一个匹配的元素。

arrayCumSum([func,] arr1, …)

返回源数组部分数据的总和，如果指定了func函数，则使用func的返回值计算总和。

示例:SELECT arrayCumSum([1, 1, 1, 1]) AS res 返回：[1,2,0,1]

arrayCumSumNonNegative(arr):与arrayCumSum相同，返回源数组部分数据的总和。不同于arrayCumSum，当返回值包含小于零的值时，该值替换为零，后续计算使用零继续计算。  
示例：SELECT arrayCumSumNonNegative([1, 1, -4, 1]) AS res 返回：[1,2,0,1]

arraySort([func,] arr1, …)

返回升序排序arr1的结果。如果指定了func函数，则排序顺序由func的结果决定。

arrayReverseSort([func,] arr1, …)

返回降序排序arr1的结果。如果指定了func函数，则排序顺序由func的结果决定。

## 时间日期函数

toYear()

toMonth()

toDayOfMonth()

toDayOfWeek()

toHour()

toMinute()

toSecond()

toMonday()

toTime()

toStartOfMonth()

toStartOfQuarter()

toStartOfYear()

toStartOfMinute()

toStartOfFiveMinute

toStartOfFifteenMinutes

toStartOfHour

toStartOfDay

timeSlot

now()

today()

yesterday()

## 条件函数

if(cond,then,else)

multiIf(cond\_1, then\_1, cond\_2, then\_2…else)

## 聚合函数

count

max

min

avg

sum

sumWithOverflow(x)：求和，如果超过上限，则报错

sumMap(key,value)：对于相同的 key，对其 value 求和

any:选择第一个遇到的值。

anyHeavy(x)：通过 [heavy hitters](https://link.hacpai.com/forward?goto=http://www.cs.umd.edu/~samir/498/karp.pdf) 算法，得到一个经常出现的值。

anyLast(x)：返回最后遇到的值

uniq(x)：求近似的排重值， 适用于数值，字符串，日期，日期时间，以及 multiple 和 tuple 参数

使用了自适合抽样算法：在 65535 下非常准确和高效。

uniqCombined(x)

求近似的排重值，适用于数值，字符串，日期，日期时间，以及 multiple 和 tuple 参数

使用了组合的三个算法：数组，hash 表和 [HyperLogLog](https://link.hacpai.com/forward?goto=https://en.wikipedia.org/wiki/HyperLogLog) 表。

内存消耗比 uniq 要小几倍，同时精度高几倍，但是性能一般比 uniq 慢，虽然有时也会快。最大支持到 96KiB 个状态。

uniqHLL12(x)

使用 HyperLogLog 算法来做排重值的近似计算，支持 2.5KB 个状态。

uniqExact(x)

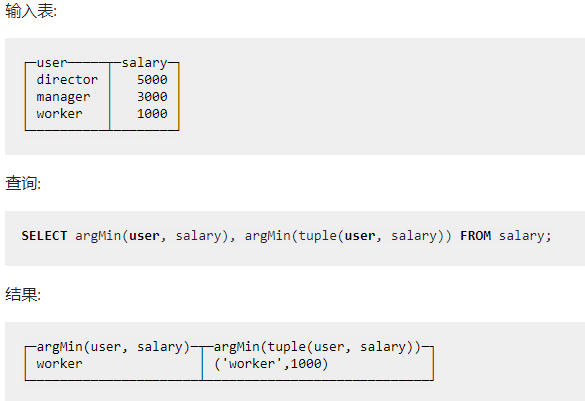
计算精确的排重值。

argMIn: argMin(arg, val) 或 argMin(tuple(arg, val))

计算 val 最小值对应的 arg 值。 如果 val 最小值存在几个不同的 arg 值，输出遇到的第一个(arg)值。

这个函数的Tuple版本将返回 val 最小值对应的tuple。本函数适合和SimpleAggregateFunction搭配使用。

示例：



argMax

topK(N)(X):返回X中出现频率最高的N个数，结果是一个数组，并且按出现频率降序排列

此函数不提供保证的结果。 在某些情况下，可能会发生错误，并且可能会返回不是最高频的值。

我们建议使用 N < 10 值，N 值越大，性能越低。最大值 N = 65536。

groupArray(x)： 创建参数值的数组。值可以按任何（不确定）顺序添加到数组中。

groupArray(max\_size)(x)：第二个版本（带有 max\_size 参数）将结果数组的大小限制为 max\_size 个元素。  
例如, groupArray (1) (x) 相当于 [any (x)] 。

在某些情况下，您仍然可以依赖执行顺序。这适用于SELECT(查询)来自使用了 ORDER BY 子查询的情况。

groupUniqArray(x)或groupUniqArray(max\_size)(x)

groupArrayInsertAt：在指定位置向数组中插入一个值。

groupArrayMovingSum(numbers\_for\_summing)

groupArrayMovingSum(window\_size)(numbers\_for\_summing)

参数：

numbers\_for\_summing — [表达式](https://clickhouse.tech/docs/zh/sql-reference/syntax/" \l "syntax-expressions) 生成数值数据类型值。

window\_size — 窗口大小。

返回值：

与输入数据大小相同的数组。  
对于输入数据类型是[Decimal](https://clickhouse.tech/docs/zh/sql-reference/data-types/decimal/) 数组元素类型是 Decimal128 。  
对于其他的数值类型, 获取其对应的 NearestFieldType 。

groupArrayMovingAvg(numbers\_for\_summing)

groupArrayMovingAvg(window\_size)(numbers\_for\_summing)

groupArraySample(max\_size)(x)：

max\_size — 结果数组的最大长度。[UInt64](https://clickhouse.tech/docs/zh/sql-reference/data-types/int-uint/)

x — 参数 (列名 或者 表达式)。

ForEach:将对 table 使用的聚合函数，转换为对数组的聚合函数。对数组的每一项进行处理，返回一个结果数组。

示例：sumForEach([1,2],[3,4,5],[6,7]) 返回：[10,13,5]

sumMap(key,value):对于相同的 key，对其 value 求和

maxMap(key,value)

minMap(key,value)

## 聚合函数组合器

-If

-If可以加到任何聚合函数之后。加了-If之后聚合函数需要接受一个额外的参数，一个条件（Uint8类型），如果条件满足，那聚合函数处理当前的行数据，如果不满足，那返回默认值（通常是0或者空字符串）。

Array

-Array后缀可以附加到任何聚合函数。 在这种情况下，聚合函数采用的参数 ‘Array(T)’ 类型（数组）而不是 ‘T’ 类型参数。 如果聚合函数接受多个参数，则它必须是长度相等的数组。 在处理数组时，聚合函数的工作方式与所有数组元素的原始聚合函数类似。

State

如果应用此combinator，则聚合函数不会返回结果值（例如唯一值的数量 [uniq](https://clickhouse.tech/docs/zh/sql-reference/aggregate-functions/reference/uniq/" \l "agg_function-uniq) 函数），但是返回聚合的中间状态（对于 uniq，返回的是计算唯一值的数量的哈希表）。 这是一个 AggregateFunction(...) 可用于进一步处理或存储在表中以完成稍后的聚合。

Merge

如果应用此组合器，则聚合函数将中间聚合状态作为参数，组合状态以完成聚合，并返回结果值。

MergeState

以与-Merge 相同的方式合并中间聚合状态。 但是，它不会返回结果值，而是返回中间聚合状态，类似于-State。

ForEach

将表的聚合函数转换为聚合相应数组项并返回结果数组的数组的聚合函数。 例如, sumForEach 对于数组 [1, 2], [3, 4, 5]和[6, 7]返回结果 [10, 13, 5] 之后将相应的数组项添加在一起。

OrDefault

更改聚合函数的行为。

如果聚合函数没有输入值，则使用此组合器它返回其返回数据类型的默认值。 适用于可以采用空输入数据的聚合函数。

-OrDefault 可与其他组合器一起使用。

OrNull

更改聚合函数的行为。

此组合器将聚合函数的结果转换为 [可为空](https://clickhouse.tech/docs/zh/sql-reference/data-types/nullable/) 数据类型。 如果聚合函数没有值来计算它返回 [NULL](https://clickhouse.tech/docs/zh/sql-reference/syntax/" \l "null-literal).

-OrNull 可与其他组合器一起使用。

Resample

允许您将数据划分为组，然后单独聚合这些组中的数据。 通过将一列中的值拆分为间隔来创建组。

## 表函数

注意：如果 **[allow\_ddl](https://clickhouse.tech/docs/zh/operations/settings/permissions-for-queries/" \l "settings_allow_ddl)** 设置被禁用，则不能使用表函数。

### file

语法：file(path, format, structure)

参数：

path — [user\_files\_path](https://clickhouse.tech/docs/zh/operations/server-configuration-parameters/settings/" \l "server_configuration_parameters-user_files_path)中文件的相对路径。在只读模式下，文件路径支持以下通配符: \*, ?, {abc,def} 和 {N..M}，其中 N, M 是数字, `'abc', 'def' 是字符串。

设置 user\_files\_path

在文件/etc/clickhouse-server/config.xml

<user\_files\_path>/var/lib/clickhouse/user\_files/</user\_files\_path>

format —文件的[格式](https://clickhouse.tech/docs/zh/interfaces/formats/" \l "formats)。

structure — 表的结构。格式 'column1\_name column1\_type, column2\_name column2\_type, ...'。

返回值

具有指定结构的表，用于读取或写入指定文件中的数据。

示例：**SELECT** \* **FROM** file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32') **LIMIT** 2;

### merge

merge(db\_name, 'tables\_regexp') – 创建一个临时Merge表。

表结构取自遇到的第一个与正则表达式匹配的表。

### numbers

numbers(N) – 返回一个包含单个 ‘number’ 列(UInt64)的表，其中包含从0到N-1的整数。  
numbers(N, M) - 返回一个包含单个 ‘number’ 列(UInt64)的表，N代表起始值，M代表连续几个

示例：-- 生成2010-01-01至2010-12-31的日期序列**select** toDate('2010-01-01') + number **as** d **FROM** numbers(365);

### url

语法

url(URL, format, structure)

参数

URL — HTTP或HTTPS服务器地址，它可以接受 GET 或 POST 请求 (对应于 SELECT 或 INSERT 查询)。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

format — 数据[格式](https://clickhouse.tech/docs/zh/interfaces/formats/" \l "formats)。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

structure — 以 'UserID UInt64, Name String' 格式的表结构。确定列名和类型。 类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

### mysql

语法：

mysql('host:port', 'database', 'table', 'user', 'password'[, replace\_query, 'on\_duplicate\_clause']);

参数：

replace\_query ：

on\_duplicate\_clause ：

### jdbc

jdbc(jdbc\_connection\_uri, schema, table) -返回通过JDBC驱动程序连接的表。

此表函数需要单独的 clickhouse-jdbc-bridge 程序才能运行。  
它支持可空类型（基于查询的远程表的DDL）。

示例：

SELECT \* FROM jdbc('jdbc:mysql://localhost:3306/?user=root&password=root', 'schema', 'table'）

### odbc

odbc(connection\_settings, external\_database, external\_table)

参数:

connection\_settings — 在 odbc.ini 文件中连接设置的部分的名称。

external\_database — 外部DBMS的数据库名。

external\_table — external\_database 数据库中的表名。

为了安全地实现ODBC连接，ClickHouse使用单独的程序 clickhouse-odbc-bridge。 如果ODBC驱动程序直接从 clickhouse-server 加载，则驱动程序问题可能会导致ClickHouse服务器崩溃。 当需要时，ClickHouse自动启动 clickhouse-odbc-bridge。 ODBC桥程序是从与 clickhouse-server 相同的软件包安装的。

外部表中字段包含的 NULL 值将转换为基本据类型的默认值。 例如，如果远程MySQL表字段包含 INT NULL 类型，则将被转换为0（ClickHouseInt32 数据类型的默认值）。

### hdfs

hdfs(URI, format, structure)

**输入参数**

URI — HDFS中文件的相对URI。

在只读模式下，文件路径支持以下通配符: \*, ?, {abc,def} 和 {N..M} ，其中 N, M 是数字, `'abc', 'def' 是字符串。

format — 文件的[格式](https://clickhouse.tech/docs/zh/interfaces/formats/" \l "formats)。

structure — 表的结构。格式 'column1\_name column1\_type, column2\_name column2\_type, ...'。

**返回值**

具有指定结构的表，用于读取或写入指定文件中的数据。

**示例**

表来自 hdfs://hdfs1:9000/test 并从中选择前两行:

SELECT \*FROM hdfs('hdfs://hdfs1:9000/test', 'TSV', 'column1 UInt32, column2 UInt32, column3 UInt32')LIMIT 2

### input

可以有效地将发送给服务器的数据转换为具有给定结构的数据并将其插入到具有其他结构的表中.

**示例：**

test 表具有以下结构 (a String, b String)

data.csv 中的数据具有不同的结构 (col1 String, col2 Date, col3 Int32)。

将数据从 data.csv 插入到 test 表中，同时进行转换的查询如下所示:

cat data.csv | clickhouse-client --query="INSERT INTO test SELECT lower(col1), col3 \* col3 FROM input('col1 String, col2 Date, col3 Int32') FORMAT CSV";

### generateRandom

生成具用给定的模式的随机数据。  
允许用数据来填充测试表。  
支持所有可以存储在表中的数据类型， LowCardinality 和 AggregateFunction除外。

generateRandom('name TypeName[, name TypeName]...', [, 'random\_seed'[, 'max\_string\_length'[, 'max\_array\_length']]]);

**参数**

name — 对应列的名称。

TypeName — 对应列的类型。

max\_array\_length — 生成数组的最大长度。 默认为10。

max\_string\_length — 生成字符串的最大长度。 默认为10。

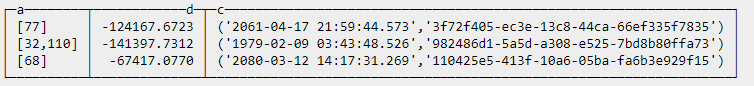
random\_seed — 手动指定随机种子以产生稳定的结果。 如果为NULL-种子是随机生成的。

**返回值**

具有请求模式的表对象。

**示例**

SELECT \* FROM generateRandom('a Array(Int8), d Decimal32(4), c Tuple(DateTime64(3), UUID)', 1, 10, 2) LIMIT 3;



### remote/remoteSecure

允许您访问远程服务器，而无需创建 Distributed 表。remoteSecure - 与 remote 相同，但是会使用加密链接。

这两个函数都可以在 SELECT 和 INSERT 查询中使用。

**语法:**

remote('addresses\_expr', db, table[, 'user'[, 'password'], sharding\_key])

remote('addresses\_expr', db.table[, 'user'[, 'password'], sharding\_key])

remoteSecure('addresses\_expr', db, table[, 'user'[, 'password'], sharding\_key])

remoteSecure('addresses\_expr', db.table[, 'user'[, 'password'], sharding\_key])

**参数**

addresses\_expr – 代表远程服务器地址的一个表达式。可以只是单个服务器地址。 服务器地址可以是 host:port 或 host。

host 可以指定为服务器名称，或是IPV4或IPV6地址。IPv6地址在方括号中指定。

port 是远程服务器上的TCP端口。 如果省略端口，则 remote 使用服务器配置文件中的 [tcp\_port](https://clickhouse.tech/docs/zh/operations/server-configuration-parameters/settings/" \l "server_configuration_parameters-tcp_port) （默认情况为，9000），remoteSecure 使用 [tcp\_port\_secure](https://clickhouse.tech/docs/zh/operations/server-configuration-parameters/settings/" \l "server_configuration_parameters-tcp_port_secure) （默认情况为，9440）。

IPv6地址需要指定端口。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

db — 数据库名。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

table — 表名。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

user — 用户名。如果未指定用户，则使用 default 。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

password — 用户密码。如果未指定密码，则使用空密码。类型: [String](https://clickhouse.tech/docs/zh/sql-reference/data-types/string/)。

sharding\_key — 分片键以支持在节点之间分发数据。

类型: [UInt32](https://clickhouse.tech/docs/zh/sql-reference/data-types/int-uint/)。

**返回值**

来自远程服务器的数据集。

**示例**:

insert into remote('127.0.0.1:9000,127.0.0.2', db, table, 'default', rand())。

### 表连接操作

INNER JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

### LIMIT操作

LIMIT N

LIMIT N BY Clause

## URL操作函数

protocol()：返回URL的协议类型

domain()：返回URL的域名

domainWithoutWWW()：返回URL不带www的域名

topLevelDomain():返回顶级域名

示例：topLevelDomain(‘http://www.baidu.com.cn’) 返回 cn

firstSignificantSubdomain()：

示例：firstSignificantSubdomain(‘http://www.baidu.com.cn’) 返回 baidu

cutToFirstSignificantSubdomain()

示例：cutToFirstSignificantSubdomain(‘http://www.baidu.com.cn’) 返回 baidu.com.cn

path()：返回URL的路径

示例：path(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split’) 返回 /s

pathFull()：返回URL的完整路径

示例：pathFull(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split’)

返回 /s?wd=SQL%E4%B8%AD%E7%9A%84split

queryString()：返回URL的参数（查询字符串）

示例：queryString(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split’)

返回 wd=SQL%E4%B8%AD%E7%9A%84split

extractURLParameters()：以列表的形式返回URL的参数

示例：extractURLParameters(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split&ur=qwguq’)

返回 [‘wd=SQL%E4%B8%AD%E7%9A%84split’,‘ur=qwguq’]

extractURLParameterNames()：以列表的形式返回URL的参数名

示例：extractURLParameterNames(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split&ur=qwguq’)

返回 [‘wd’,‘ur’]

cutQueryString()：返回URL？（参数）前面的内容

示例：cutQueryString(‘https://www.baidu.com/s?wd=SQL%E4%B8%AD%E7%9A%84split&ur=qwguq’)

返回 https://www.baidu.com/s

## IP操作函数

IPv4StringToNum(s)：将IPV4转为数值，非IPV4的转化为0

IPv4NumToString(num)：将数值转为IPV4

示例：IPv4NumToString(400148037) 返回 23.217.198.69

IPv4NumToStringClassC(num)：将数值转为IPV4，且最后的段位用xxx代替

示例：IPv4NumToStringClassC(400148037) 返回 23.217.198.xxx

## 特别

alphaTokens(s)

从a-z和A-Z中返回连续字节

示例：SELECT alphaTokens('abca1abc')  返回： ['abca','abc']

# 字典

## 介绍

字典是一个映射 (键 -> 属性）, 是方便各种类型的参考清单。

ClickHouse支持一些特殊函数配合字典在查询中使用。 将字典与函数结合使用比将 JOIN 操作与引用表结合使用更简单、更有效。

[NULL](https://clickhouse.tech/docs/zh/sql-reference/syntax/" \l "null-literal) 值不能存储在字典中。

ClickHouse支持:

[内置字典](https://clickhouse.tech/docs/zh/sql-reference/dictionaries/internal-dicts/" \l "internal_dicts) ,这些字典具有特定的 [函数集](https://clickhouse.tech/docs/zh/sql-reference/functions/ym-dict-functions/).

[插件（外部）字典](https://clickhouse.tech/docs/zh/sql-reference/dictionaries/external-dictionaries/external-dicts/" \l "dicts-external-dicts) ,这些字典拥有一个 [函数集](https://clickhouse.tech/docs/zh/sql-reference/functions/ext-dict-functions/).

## 外部字典

### 配置

**使用xml文件配置字典**

<dictionary>

<name>dict\_name</name>

<structure>

<!-- Complex key configuration -->

</structure>

<source>

<!-- Source configuration -->

</source>

<layout>

<!-- Memory layout configuration -->

</layout>

<lifetime>

<!-- Lifetime of dictionary in memory -->

</lifetime></dictionary>

**示例**

<dictionaries>

<dictionary>

<name>test</name>

<source>

<odbc>

<connection\_string>DSN=pg;UID=;PWD=;HOST=;PORT=5432;DATABASE=</connection\_string>

<table>product</table>

</odbc>

</source>

<lifetime>

<min>300</min>

<max>360</max>

</lifetime>

<layout>

<hashed/>

</layout>

<structure>

<id>

<name>id</name>

</id>

<attribute>

<name>del\_flag</name>

<type>UInt64</type>

<null\_value>0</null\_value>

</attribute>

</structure>

</dictionary>

</dictionaries>

**使用DDL语句**

CREATE DICTIONARY dict\_name

(

... -- attributes

)

PRIMARY KEY ... -- complex or single key configuration

SOURCE(...) -- Source configuration

LAYOUT(...) -- Memory layout configuration

LIFETIME(...) -- Lifetime of dictionary in memory

### 来源

#### 本地文件

设置示例:

<source>

<file>

<path>/opt/dictionaries/os.tsv</path>

<format>TabSeparated</format>

</file>

</source>

或

SOURCE(FILE(path './user\_files/os.tsv' format 'TabSeparated'))

#### 可执行文件

使用可执行文件取决于 [字典如何存储在内存中](https://clickhouse.tech/docs/zh/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-layout/). 如果字典存储使用 cache 和 complex\_key\_cache，ClickHouse通过向可执行文件的STDIN发送请求来请求必要的密钥。 否则，ClickHouse将启动可执行文件并将其输出视为字典数据。

设置示例:

<source>

<executable>

<command>cat /opt/dictionaries/os.tsv</command>

<format>TabSeparated</format>

</executable>

</source>

或

SOURCE(EXECUTABLE(command 'cat /opt/dictionaries/os.tsv' format 'TabSeparated'))

#### Http(s)

使用HTTP（s）服务器取决于 [字典如何存储在内存中](https://clickhouse.tech/docs/zh/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-layout/). 如果字典存储使用 cache 和 complex\_key\_cache，ClickHouse通过通过发送请求请求必要的密钥 POST 方法。

设置示例:

<source>

<http>

<url>http://[::1]/os.tsv</url>

<format>TabSeparated</format>

<credentials>

<user>user</user>

<password>password</password>

</credentials>

<headers>

<header>

<name>API-KEY</name>

<value>key</value>

</header>

</headers>

</http>

</source>

或

SOURCE(HTTP(

url 'http://[::1]/os.tsv'

format 'TabSeparated'

credentials(user 'user' password 'password')

headers(header(name 'API-KEY' value 'key'))))

为了让ClickHouse访问HTTPS资源，您必须 [配置openSSL](https://clickhouse.tech/docs/zh/operations/server-configuration-parameters/settings/" \l "server_configuration_parameters-openssl) 在服务器配置中。

设置字段:

url – The source URL.

format – The file format. All the formats described in “[格式](https://clickhouse.tech/docs/zh/interfaces/formats/" \l "formats)” 支持。

credentials资格证书 – 基本HTTP身份验证。可选参数。

user – Username required for the authentication.

password – Password required for the authentication.

headers – All custom HTTP headers entries used for the HTTP request. Optional parameter.

header – Single HTTP header entry.

name – Identifiant name used for the header send on the request.

value – Value set for a specific identifiant name.

#### ODBC

**设置示例**

<source>

<odbc>

<db>DatabaseName</db>

<table>ShemaName.TableName</table>

<connection\_string>DSN=some\_parameters</connection\_string>

<invalidate\_query>SQL\_QUERY</invalidate\_query>

</odbc>

</source>

或

**SOURCE**(ODBC(

db 'DatabaseName'

**table** 'SchemaName.TableName'

connection\_string 'DSN=some\_parameters'

invalidate\_query 'SQL\_QUERY'))

#### DBMS

##### mysql

**连接示例**

**SOURCE**(MYSQL(

port 3306

**user** 'clickhouse'

password 'qwerty'

replica(**host** 'example01-1' priority 1)

replica(**host** 'example01-2' priority 1)

db 'db\_name'

**table** 'table\_name'

**where** 'id=10'

invalidate\_query 'SQL\_QUERY'))

##### clickhouse

**SOURCE**(CLICKHOUSE(

**host** 'example01-01-1'

port 9000

**user** 'default'

password ''

db 'default'

**table** 'ids'

**where** 'id=10'))

##### MongoDB

**SOURCE**(MONGO(

**host** 'localhost'

port 27017

**user** ''

password ''

db 'test'

collection 'dictionary\_source'))

##### Redis

**SOURCE**(REDIS(

**host** 'localhost'

port 6379

storage\_type 'simple'

db\_index 0))

### 更新

配置更新间隔

<dictionary>

...

<lifetime>300</lifetime>

...

</dictionary>

或

CREATE DICTIONARY (...)...LIFETIME(300)...

设置为 <lifetime>0</lifetime> (LIFETIME(0)）可以防止字典更新。

**设置一个时间间隔**，ClickHouse将在此范围内选择一个统一的随机时间。 为了在大量服务器上升级时分配字典源上的负载，这是必要的。

<dictionary>

...

<lifetime>

<min>300</min>

<max>360</max>

</lifetime>

...

</dictionary>

或

LIFETIME(MIN 300 MAX 360)

升级字典时，ClickHouse服务器根据字典的类型应用不同的逻辑:

对于文本文件，它检查修改的时间。 如果时间与先前记录的时间不同，则更新字典。

对于MyISAM表，修改的时间使用检查 SHOW TABLE STATUS 查询。

默认情况下，每次都会更新来自其他来源的字典。

对于MySQL（InnoDB），ODBC和ClickHouse源代码，您可以设置一个查询，只有在字典真正改变时才会更新字典，而不是每次都更新。 为此，请按照下列步骤操作:

字典表必须具有在源数据更新时始终更改的字段。

源的设置必须指定检索更改字段的查询。 ClickHouse服务器将查询结果解释为一行，如果此行相对于其以前的状态发生了更改，则更新字典。 指定查询 <invalidate\_query> 字段中的设置 [来源](https://clickhouse.tech/docs/zh/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-sources/).

设置示例:

<dictionary>

...

<odbc>

...

<invalidate\_query>SELECT update\_time FROM dictionary\_source where id = 1</invalidate\_query>

</odbc>

...

</dictionary>

或

...

SOURCE(ODBC(... invalidate\_query 'SELECT update\_time FROM dictionary\_source where id = 1'))

...

### 字典键和字段

##### **键**

**数字键**

类型: UInt64.

配置示例

<id>

<name>Id</name>

</id>

或

**CREATE** **DICTIONARY** (

Id UInt64,

...)**PRIMARY** **KEY** Id

复合密钥

类型：tuple

配置示例

<structure>

<key>

<attribute>

<name>field1</name>

<type>String</type>

</attribute>

<attribute>

<name>field2</name>

<type>UInt32</type>

</attribute>

...

</key>

...

或

**CREATE** **DICTIONARY** (

field1 String,

field2 String

...)**PRIMARY** **KEY** field1, field2...

##### 字段

配置示例

<structure>

...

<attribute>

<name>Name</name>

<type>ClickHouseDataType</type>

<null\_value></null\_value>

<expression>rand64()</expression>

<hierarchical>true</hierarchical>

<injective>true</injective>

<is\_object\_id>true</is\_object\_id>

</attribute>

</structure>

或

**CREATE** **DICTIONARY** somename (

Name ClickHouseDataType **DEFAULT** '' EXPRESSION rand64() HIERARCHICAL INJECTIVE IS\_OBJECT\_ID)

### 分层字典

**配置示例**

<dictionary>

<structure>

<id>

<name>region\_id</name>

</id>

<attribute>

<name>parent\_region</name>

<type>UInt64</type>

<null\_value>0</null\_value>

<hierarchical>true</hierarchical>

</attribute>

<attribute>

<name>region\_name</name>

<type>String</type>

<null\_value></null\_value>

</attribute>

</structure>

</dictionary>

parent\_region：父项键

**示例**

1.创建表和数据，创建字典

drop table t\_region;

create table t\_region(region\_id UInt64, parent\_region UInt64, region\_name String) ENGINE=TinyLog;

insert into t\_region values

(1, 0, 'jiangsu'),(2, 1, 'suzhou'),(3, 2, 'huqiu'),(4, 0, 'anhui'),(5, 4, 'hefei');

创建字典， 指定HIERARCHICAL字段：

DROP DICTIONARY t\_dict\_region;

CREATE DICTIONARY t\_dict\_region (

region\_id UInt64,

parent\_region UInt64 HIERARCHICAL,

region\_name String

)

PRIMARY KEY region\_id

SOURCE(CLICKHOUSE(

host 'localhost'

port 9001

user 'default'

db 'default'

password ''

table 't\_region'

))

LAYOUT(HASHED())

LIFETIME(30);

2.字典的查询

SELECT dictGetString('default.t\_dict\_region', 'region\_name', toUInt64(2)) AS regionName;

┌─regionName─┐

│ suzhou │

└────────────┘

SELECT dictGetHierarchy('default.t\_dict\_region', toUInt64(3));

┌─dictGetHierarchy('default.t\_dict\_region', toUInt64(3))─┐

│ [3,2,1] │

└────────────────────────────────────────────────────────┘

## 内部字典

地理字典

ClickHouse包含用于处理地理数据库的内置功能。

默认禁用内部字典。  
启用：

配置文件：config.xml

取消注释参数 path\_to\_regions\_hierarchy\_file 和 path\_to\_regions\_names\_files

# 操作符

## in操作符

IN和GLOBAL IN区别（还有NOT IN 和 GLOBAL NOT IN）

在集群查询中，查询中有子查询，并且子查询用的是分布式表，则使用GLOBAL IN

示例：

**SELECT** uniq(UserID) **FROM** distributed\_table **WHERE** CounterID = 101500 **AND** UserID **GLOBAL** **IN** (**SELECT** UserID **FROM** distributed\_table **WHERE** CounterID = 34)

使用GLOBAL IN 会先运行子查询，并将查询结果放在内存中的临时表中，然后在把结果发送给每个远程服务器。

这比使用正常IN更优化。 但是，注意以下几点:

1. 创建临时表时，数据有重复，请在子查询中指定DISTINCT，以此来减少通过网络传输的数据量。
2. 临时表将被发送到所有远程服务器。 传输不考虑网络拓扑。 例如，如果10个远程服务器驻留在与请求者服务器非常远程的数据中心中，则数据将通过通道发送10次到远程数据中心。 使用GLOBAL IN时尽量避免使用大型数据集。
3. 将数据传输到远程服务器时，无法配置网络带宽限制。 您可能会使网络过载。
4. 尝试跨服务器分发数据，以便您不需要定期使用GLOBAL IN。
5. 如果您需要经常使用GLOBAL IN，请规划ClickHouse集群的位置，以便单个副本组驻留在不超过一个数据中心中，并且它们之间具有快速网络，以便可以完全在单个数据中心内处理查询。

# 操作

## 要求

### CPU

具有x86\_64架构并支持SSE4.2指令的CPU

选择处理器时核数多更重要

### RAM

内存要求不高，需要多少内存取决于：查询的复杂性和查询中处理的数据量。

要计算所需的RAM体积，您应该估计临时数据的大小 [GROUP BY](https://clickhouse.tech/docs/zh/sql-reference/statements/select/group-by/" \l "select-group-by-clause), [DISTINCT](https://clickhouse.tech/docs/zh/sql-reference/statements/select/distinct/" \l "select-distinct), [JOIN](https://clickhouse.tech/docs/zh/sql-reference/statements/select/join/" \l "select-join) 和您使用的其他操作。

ClickHouse可以使用外部存储器来存储临时数据，就是内存不足时将临时数据转存到磁盘上

**设置示例：在users.xml文件中**

set max\_bytes\_before\_external\_group\_by=20000000000; #20G，默认值为0，表示在GROUP BY中启动将临时数据转存到磁盘上的内存阈值。

# 使用max\_bytes\_before\_external\_group\_by时，建议将max\_memory\_usage设置为它的两倍。这是因为一个聚合需要两个阶段来完成： #（1）读取数据并形成中间数据 #（2）合并中间数据。临时数据的转存只会发生在第一个阶段。 # 如果没有发生临时文件的转存，那么阶段二将最多消耗与1阶段相同的内存大小。 set max\_memory\_usage=40000000000; #40G

### 交换文件

禁用生产环境的交换文件。

禁用交换内存：

echo "vm.swappiness = 1" >> /etc/sysctl.conf

echo 1 > /proc/sys/vm/swappiness

### 存储

**估计数据量。**

您可以采取数据的样本并从中获取行的平均大小。 然后将该值乘以计划存储的行数。

**估计数据压缩系数。**

请将数据的样本加载到ClickHouse中，并将数据的实际大小与存储的表的大小进行比较。

### 网络

如果可能的话，使用10G或更高级别的网络。

### 软件

ClickHouse主要是为Linux系列操作系统开发的。 推荐的Linux发行版是Ubuntu。

## 监控

### 硬件资源利用率

处理器上的负载和温度：可以使用[dmesg](https://en.wikipedia.org/wiki/Dmesg" \t "https://clickhouse.tech/docs/zh/operations/monitoring/_blank), [turbostat](https://www.linux.org/docs/man8/turbostat.html" \t "https://clickhouse.tech/docs/zh/operations/monitoring/_blank)或者其他工具。

磁盘存储，RAM，网络的使用率

### ClickHouse 服务的指标

1. 要跟踪服务器事件，请观察服务器日志。
2. 可以在[系统指标](https://clickhouse.tech/docs/zh/operations/system-tables/metrics/" \l "system_tables-metrics)，[系统事件](https://clickhouse.tech/docs/zh/operations/system-tables/events/" \l "system_tables-events)以及[系统异步指标](https://clickhouse.tech/docs/zh/operations/system-tables/asynchronous_metrics/" \l "system_tables-asynchronous_metrics)等系统表查看所有的指标项。
3. 可以配置ClickHouse向[Graphite](https://github.com/graphite-project" \t "https://clickhouse.tech/docs/zh/operations/monitoring/_blank)推送监控信息并导入指标

## 更新

如果从deb包安装ClickHouse，请在服务器上执行以下命令:

sudo apt-get update

$ sudo apt-get install clickhouse-client clickhouse-server

$ sudo service clickhouse-server restart

ClickHouse不支持分布式更新。该操作应在每个单独的服务器上连续执行。不要同时更新群集上的所有服务器，否则群集将在一段时间内不可用。

## 数据备份

1.副本备份

2.不能人工删除使用带有MergeTree引擎且包含超过50Gb数据的表：max-table-size-to-drop

3.将源数据复制到其它地方

例如数据源为kafka时，可以额外配置一组订阅服务器，这些订阅服务器将在写入ClickHouse时读取相同的数据流，并将其存储在冷存储中。 大多数公司已经有一些默认推荐的冷存储，可能是对象存储或分布式文件系统，如 [HDFS]

4.clickhouse-copier

[clickhouse-copier] (utilities/clickhouse-copier.md) 是一个多功能工具，最初创建它是为了用于重新切分pb大小的表。 因为它能够在ClickHouse表和集群之间可靠地复制数据，所以它也可用于备份和还原数据。

## 使用建议

**CPU**

建议始终使用 performance 频率调节器，on-demand 频率调节器在持续高需求的情况下，效果比较差。

performance-高性能模式:在这个模式系统会按设定最大主频率满负荷运转，主频会一直保持在设定范围内的最大值。

ondemand-按需响应模式：这个模式一般是系统默认模式，根据需要自动调节cpu频率，此模式的特点是频率升高需条件触发，反应迅速，频率下降无需触发，不需要高频率时自动渐渐下降。

**大页(Huge Pages)**

始终禁用透明大页(transparent huge pages)。 它会干扰内存分配器，从而导致显着的性能下降。

echo 'never' | sudo tee /sys/kernel/mm/transparent\_hugepage/enabled

**磁盘**

优先SSD，如果没有，请使用硬盘。 SATA硬盘7200转就行了。

**Linux内核**

不要使用过时的Linux内核

**虚拟机监视器(Hypervisor)配置**

如果您使用的是OpenStack，请在nova.conf中设置

cpu\_mode=host-passthrough

如果您使用的是libvirt，请在XML配置中设置

<cpu mode='host-passthrough'/>

这对于ClickHouse能够通过 cpuid 指令获取正确的信息非常重要。  
否则，当在旧的CPU型号上运行虚拟机监视器时，可能会导致 Illegal instruction 崩溃。

# 异常

## 内存限制

ClickHouse 查询时，报错：Memory limit (for query) exceeded: would use 9.38 GiB (attempt to allocate chunk of 135439453 bytes), maximum: 9.31 GiB

**报错原因**

默认情况下，ClickHouse会限制group by使用的内存量（它使用 hash table来处理group by）。

**解决方法**

1. 内存足够情况下

SET max\_memory\_usage = 128000000000; #128G

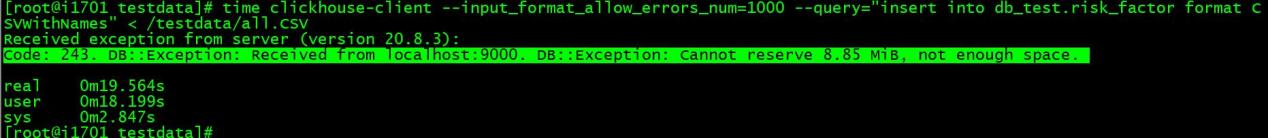
2. 内存不足情况下

GROUP BY中允许将临时数据转存到磁盘上，以限制对内存的使用

set max\_bytes\_before\_external\_group\_by=20000000000; #20G，默认值为0，表示在GROUP BY中启动将临时数据转存到磁盘上的内存阈值。

# 使用max\_bytes\_before\_external\_group\_by时，建议将max\_memory\_usage设置为它的两倍。这是因为一个聚合需要两个阶段来完成：#（1）读取数据并形成中间数据 #（2）合并中间数据。临时数据的转存只会发生在第一个阶段。# 如果没有发生临时文件的转存，那么阶段二将最多消耗与1阶段相同的内存大小。set max\_memory\_usage=40000000000; #40G

## 存储空间不足



**解决方法**

1.在clickhouse安装时，启动之前把config.xml中的path路径改为一个有足够空间的路径。

2.如果使用一段时间发现空间不够了，可以使用软连接，将/data数据放到一个有足够空间的目录下。

graphite 监控

## 字段为空

DB::Exception: Cannot convert NULL value to non-Nullable type: while converting source column second\_channel to destination column second\_channel (version 20.4.6.53 (official build))

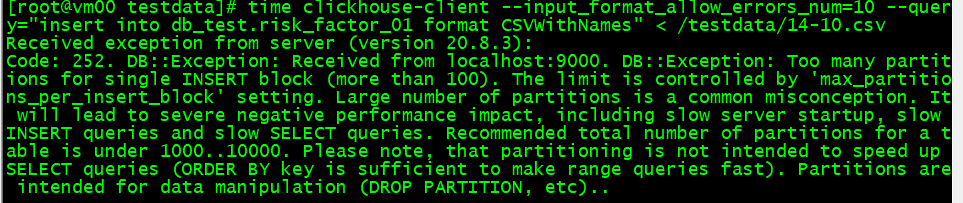
**报错原因**

字段类型是非空类型，insert null值到非空字段second\_channel会报错。

**解决方法**

可以将非空类型改成Nullable(String)，但是要注意Nullable字段不允许用于order by。

## 分区数量限制



**报错原因**

clickhouse默认一次数据插入最多生成分区不能超过100个，如果生成分区超过100个就会报此异常

**解决方法**

在users.xml 文件中加入

<max\_partitions\_per\_insert\_block>1000</max\_partitions\_per\_insert\_block>

注意clickhouse的分区一般不要超过1000，分区过多会降低查询速度。