Clickhouse

版本：V 1.0.5

# 第一章 Clickhouse简介

ClickHouse 是俄罗斯的Yandex于2016年开源的列式存储数据库（DBMS），使用C++语言编写，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。

# 第二章 Clickhouse的特点

## 2.1 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

**列式储存的好处：**

1. 对于列的聚合，计数，求和等统计操作原因优于行式存储。
2. 由于某一列的数据类型都是相同的，针对于数据存储更容易进行数据压缩，每一列选择更优的数据压缩算法，大大提高了数据的压缩比重。

3 由于数据压缩比更好，一方面节省了磁盘空间，另一方面对于cache也有了更大的发挥空间。

## DBMS的功能

几乎覆盖了标准SQL的大部分语法，包括 DDL和 DML ,以及配套的各种函数。

用户管理及权限管理

数据的备份与恢复

## 2.3 多样化引擎

clickhouse和mysql类似，把表级的存储引擎插件化，根据表的不同需求可以设定不同的存储引擎。目前包括合并树、日志、接口和其他四大类20多种引擎。

## 2.4 高吞吐写入能力

ClickHouse采用类LSM Tree的结构，数据写入后定期在后台Compaction。通过类LSM tree的结构，ClickHouse在数据导入时全部是顺序append写，写入后数据段不可更改，在后台compaction时也是多个段merge sort后顺序写回磁盘。顺序写的特性，充分利用了磁盘的吞吐能力，即便在HDD上也有着优异的写入性能。

官方公开benchmark测试显示能够达到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。

## 2.5 数据分区与线程级并行

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。

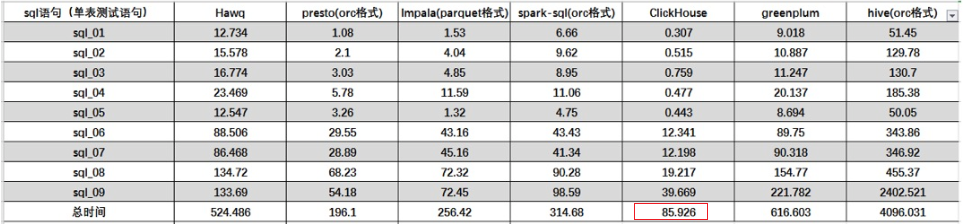
在这种设计下，单条Query就能利用整机所有CPU。极致的并行处理能力，极大的降低了查询延时。

所以，clickhouse即使对于大量数据的查询也能够化整为零平行处理。但是有一个弊端就是对于单条查询使用多cpu，就不利于同时并发多条查询。所以对于高qps的查询业务，clickhouse并不是强项。

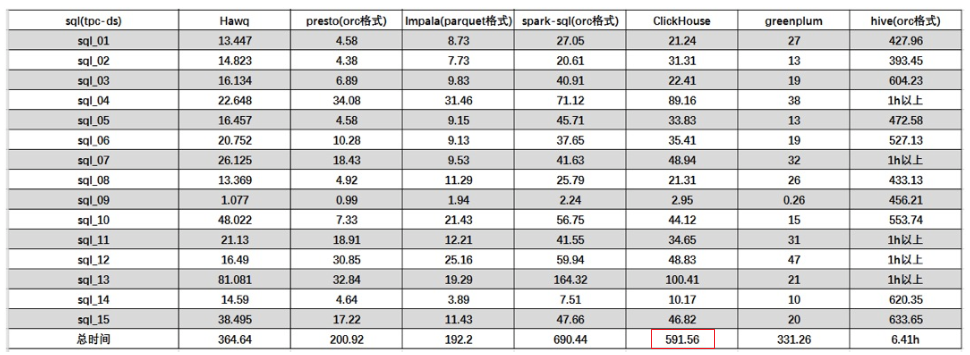
## 2.6 性能对比

某网站精华帖，中对几款数据库做了性能对比。

单表查询：



关联查询：



结论: clickhouse像很多OLAP数据库一样，单表查询速度由于关联查询，而且clickhouse的两者差距更为明显。

# 第三章 clickhouse的安装

## 3.1 准备工作

### 3.1.1 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 ~]$ vim /etc/security/limits.conf

在文件末尾添加：

\* soft nofile 65536

\* hard nofile 65536

\* soft nproc 131072

\* hard nproc 131072

3.1.2 CentOS取消SELINUX

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

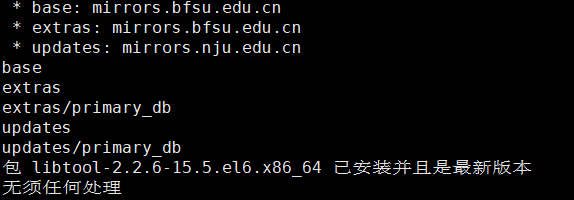
[root@hadoop102 ~]# vim /etc/selinux/config

SELINUX=disabled

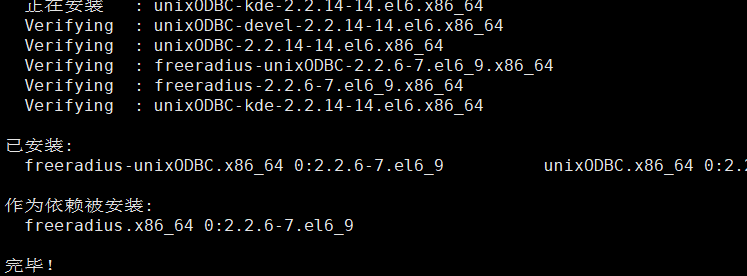
3.1.3 关闭防火墙

3.1.4 安装依赖

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



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



## 3.2 单机安装

官网：https://clickhouse.yandex/

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

3.2.1 上传4个文件

[root@hadoop102 software]# ls

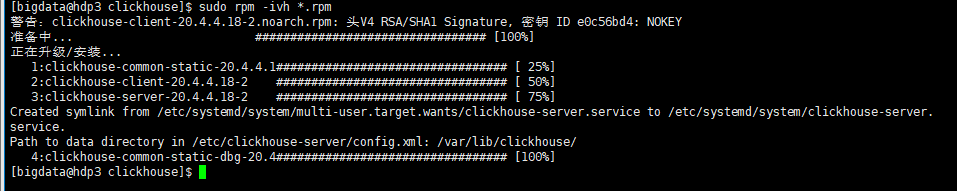
clickhouse/clickhouse-client-20.4.4.18-2.noarch.rpm

clickhouse/clickhouse-common-static-20.4.4.18-2.x86\_64.rpm

clickhouse/clickhouse-common-static-dbg-20.4.4.18-2.x86\_64.rpm

clickhouse/clickhouse-server-20.4.4.18-2.noarch.rpm

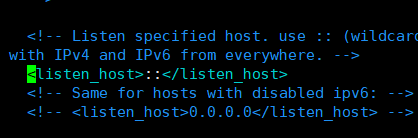
3.2.2分别在三台机子上安装这4个rpm文件



3.2.3 修改配置文件：

[bigdata@hdp1 soft]# sudo vim /etc/clickhouse-server/config.xml

把 **<listen\_host>::</listen\_host>** 的注解打开，这样的话才能让clickhouse被除本机以外的服务器访问



3.2.4 启动ClickServer

[bigdata@hdp1 soft]#sudo systemctl start clickhouse-server

3.2.5 使用client连接server

[bigdata@hdp1 soft]# clickhouse-client -m

### 3.2.6关闭开机自启

sudo systemctl disable clickhouse-server



第四章 数据类型

4.1 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]

**使用场景： 个数、数量、也可以存储型id。**

4.2 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 │

└─────────────────────┘

**使用场景：一般数据值比较小，不涉及大量的统计计算，精度要求不高的时候。比如保存商品的重量。**

## 4.3 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。

## 4.4 Decimal 型

有符号的浮点点数，可在加、减和乘法运算过程中保持精度。对于除法，最低有效数字会被丢弃（不舍入）。

有三种声明：Decimal32(s)，相当于Decimal(9-s,s)

Decimal64(s)，相当于Decimal(18-s,s)

Decimal128(s)，相当于Decimal(38-s,s)

s标识小数位

**使用场景： 一般金额字段、汇率、利率等字段为了保证小数点精度，都使用Decimal进行存储。**

4.4 字符串

1）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。

2）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。

**使用场景：名称、文字描述、字符型编码。 固定长度的可以保存一些定长的内容，比如一些编码，性别等但是考虑到一定的变化风险，带来收益不够明显，所以定长字符串使用意义有限。**

4.5 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 │

└─────────────────┘

**使用场景：对一些状态、类型的字段算是一种空间优化，也算是一种数据约束。但是实际使用中往往因为一些数据内容的变化增加一定的维护成本，甚至是数据丢失问题。所以谨慎使用。**

4.8 时间类型

目前clickhouse 有三种时间类型

Date 接受 **年-月-日** 的字符串比如 ‘2019-12-16’

Datetime 接受 **年-月-日 时:分:秒** 的字符串比如 ‘2019-12-16 20:50:10’

Datetime64 接受 **年-月-日 时:分:秒.亚秒** 的字符串比如 ‘2019-12-16 20:50:10.66’

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/>

4.9 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.

第五章 表引擎

5.1 表引擎的使用

表引擎是clickhouse的一大特色。可以说， 表引擎决定了如何存储标的数据。包括：

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。

表引擎的使用方式就是必须显形在创建表时定义该表使用的引擎，以及引擎使用的相关参数。如：

create table t\_tinylog ( id String, name String) **engine=TinyLog**;

**特别注意：引擎的名称大小写敏感**

## 5.2 TinyLog

以列文件的形式保存在磁盘上，不支持索引，没有并发控制。一般保存少量数据的小表，生产环境上作用有限。可以用于平时练习测试用。

5.3 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。

## 5.4 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。地位可以相当于innodb之于Mysql。 而且基于MergeTree，还衍生除了很多小弟，也是非常有特色的引擎。

建表语句

create table t\_order\_mt(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime

) engine =**MergeTree**

**partition by** toYYYYMMDD(create\_time)

**primary key** (id)

**order** **by** (id,sku\_id)

insert into t\_order\_mt

values(101,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

(102,'sku\_002',2000.00,'2020-06-01 11:00:00'),

(102,'sku\_004',2500.00,'2020-06-01 12:00:00'),

(102,'sku\_002',2000.00,'2020-06-01 13:00:00')

(102,'sku\_002',12000.00,'2020-06-01 13:00:00')

(102,'sku\_002',600.00,'2020-06-02 12:00:00')

MergeTree其实还有很多参数(绝大多数用默认值即可)，但是三个参数是更加重要的，也涉及了关于MergeTree的很多概念。

### 5.4.1 partition by 分区 （可选项）

**作用：** 学过hive的应该都不陌生，分区的目的主要是降低扫描的范围，优化查询速度。

**如果不填：** 只会使用一个分区。

**分区目录：** MergeTree 是以列文件+索引文件+表定义文件组成的，但是如果设定了分区那么这些文件就会保存到不同的分区目录中。

**并行：**分区后，面对涉及跨分区的查询统计，clickhouse会以分区为单位并行处理。

**数据写入与分区合并：**

任何一个批次的数据写入都会产生一个临时分区，不会纳入任何一个已有的分区。写入后的某个时刻（大概10-15分钟后），clickhouse会自动执行合并操作（等不及也可以手动通过optimize执行），把临时分区的数据，合并到已有分区中。

optimize table xxxx [final]

### 5.4.2 primary key主键(可选)

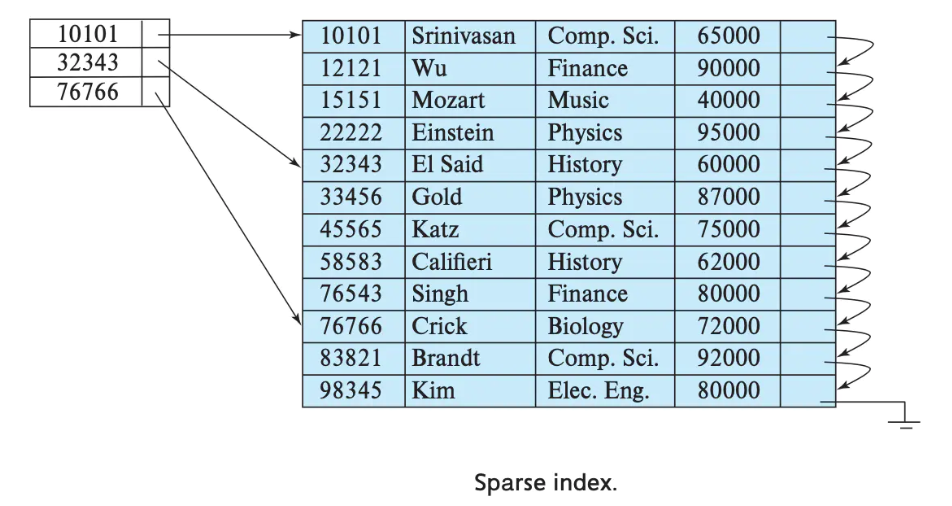
clickhouse中的主键，和其他数据库不太一样，**它只提供了数据的一级索引，但是却不是唯一约束**。这就意味着是可以存在相同primary key的数据的。

主键的设定主要依据是查询语句中的where 条件。

根据条件通过对主键进行某种形式的二分查找，能够定位到对应的index granularity,避免了全包扫描。

index granularity： 直接翻译的话就是索引粒度，指在稀疏索引中两个相邻索引对应数据的间隔。clickhouse中的MergeTree默认是8192。官方不建议修改这个值，除非该列存在大量重复值，比如在一个分区中几万行才有一个不同数据。

稀疏索引：



稀疏索引的好处就是可以用很少的索引数据，定位更多的数据，代价就是只能定位到索引粒度的第一行，然后再进行进行一点扫描。

### 5.4.3 order by （必选）

order by 设定了**分区内**的数据按照哪些字段顺序进行有序保存。

order by是MergeTree中唯一一个必填项，甚至比primary key 还重要，因为当用户不设置主键的情况，很多处理会依照order by的字段进行处理（比如后面会讲的去重和汇总）。

**要求：主键必须是order by字段的前缀字段。**

比如order by 字段是 (id,sku\_id) 那么主键必须是id 或者(id,sku\_id)

### 5.4.4 二级索引

目前在clickhouse的官网上二级索引的功能是被标注为**实验性**的。

所以使用二级索引前需要增加设置

set allow\_experimental\_data\_skipping\_indices=1;

create table t\_order\_mt2(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime,

**INDEX a total\_amount TYPE minmax GRANULARITY 5**

) engine =MergeTree

partition by toYYYYMMDD(create\_time)

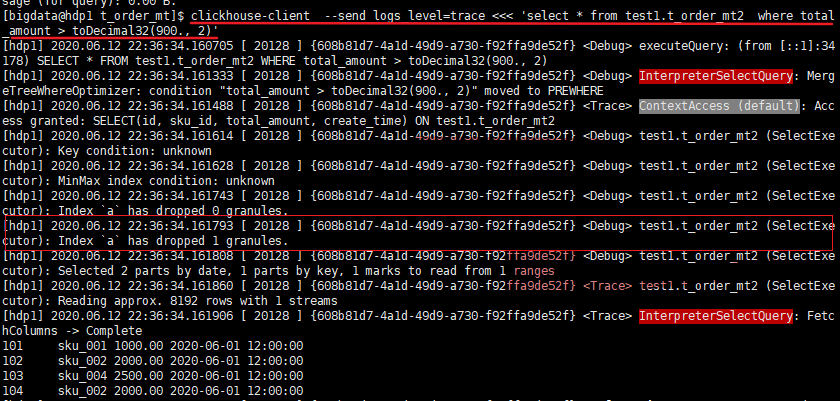
primary key (id)

order by (id, sku\_id)

其中**GRANULARITY N** 是设定二级索引对于一级索引粒度的粒度。

那么在使用下面语句进行测试，可以看出二级索引能够为非主键字段的查询发挥作用。

[bigdata@hdp1 t\_order\_mt]$ clickhouse-client --send\_logs\_level=trace <<< 'select \* from test1.t\_order\_mt where total\_amount > toDecimal32(900., 2)'



### 5.4.5 数据TTL

TTL即Time To Live，MergeTree提供了可以管理数据或者列的生命周期的功能。

列级别TTL

create table t\_order\_mt3(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2) TTL create\_time+interval 10 SECOND,

create\_time Datetime

) engine =MergeTree

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id, sku\_id)

插入数据

insert into t\_order\_mt3

values(106,'sku\_001',1000.00,'2020-06-12 22:52:30') ,

(107,'sku\_002',2000.00,'2020-06-12 22:52:30'),

(110,'sku\_003',600.00,'2020-06-13 12:00:00')

表级TTL

针对整张表

下面的这条语句是数据会在create\_time 之后10秒丢失

alter table t\_order\_mt3 MODIFY TTL create\_time + INTERVAL 10 SECOND;

涉及判断的字段必须是Date或者Datetime类型，推荐使用分区的日期字段。

能够使用的时间周期：

- SECOND

- MINUTE

- HOUR

- DAY

- WEEK

- MONTH

- QUARTER

- YEAR

## 5.5 ReplacingMergeTree

ReplacingMergeTree是MergeTree的一个变种，它存储特性完全继承MergeTree，只是多了一个去重的功能。

尽管MergeTree可以设置主键，但是primary key其实没有唯一约束的功能。如果你想处理掉重复的数据，可以借助这个ReplacingMergeTree。

**去重时机**：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。

**去重范围**：如果表经过了分区，去重只会在分区内部进行去重，不能执行跨分区的去重。

所以ReplacingMergeTree能力有限， 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。

create table t\_order\_rmt(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2) ,

create\_time Datetime

) engine =ReplacingMergeTree(create\_time)

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id, sku\_id)

**ReplacingMergeTree() 填入的参数为版本字段，重复数据保留版本字段值最大的。**

**如果不填版本字段，默认保留最后一条。**

insert into t\_order\_rmt

values(101,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

(102,'sku\_002',2000.00,'2020-06-01 11:00:00'),

(102,'sku\_004',2500.00,'2020-06-01 12:00:00'),

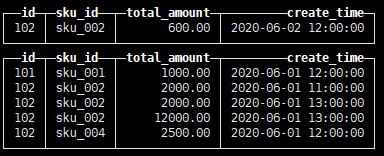
(102,'sku\_002',2000.00,'2020-06-01 13:00:00')

(102,'sku\_002',12000.00,'2020-06-01 13:00:00')

(102,'sku\_002',600.00,'2020-06-02 12:00:00')

SELECT \*

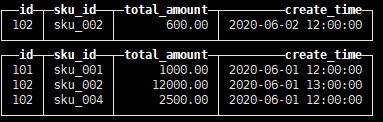
FROM t\_order\_rmt



OPTIMIZE TABLE t\_order\_rmt FINAL

SELECT \*

FROM t\_order\_rmt



通过测试得到结论：

* 实际上是使用order by 字段作为唯一键。
* 去重不能跨分区。
* 只有合并分区才会进行去重。
* 认定重复的数据保留，版本字段值最大的。
* 如果版本字段相同则保留最后一笔。

## 5.6 SummingMergeTree

对于不查询明细，只关心以维度进行汇总聚合结果的场景。如果只使用普通的MergeTree的话，无论是存储空间的开销，还是查询时临时聚合的开销都比较大。

Clickhouse 为了这种场景，提供了一种能够“预聚合”的引擎，SummingMergeTree.

表定义

create table t\_order\_smt(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2) ,

create\_time Datetime

) engine =SummingMergeTree(total\_amount)

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id,sku\_id )

插入数据

insert into t\_order\_smt

values(101,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

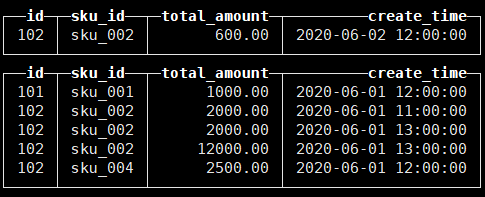
(102,'sku\_002',2000.00,'2020-06-01 11:00:00'),

(102,'sku\_004',2500.00,'2020-06-01 12:00:00'),

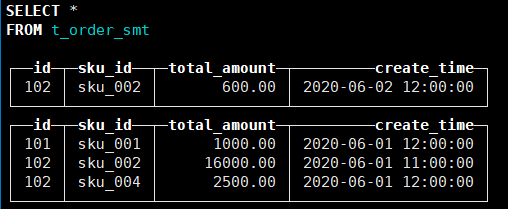
(102,'sku\_002',2000.00,'2020-06-01 13:00:00')

(102,'sku\_002',12000.00,'2020-06-01 13:00:00')

(102,'sku\_002',600.00,'2020-06-02 12:00:00')



optimize table t\_order\_smt final;



通过结果可以得到以下结论：

以SummingMergeTree（）中指定的列作为汇总数据列。可以填写多列必须数字列，如果不填，以所有非维度列且为数字列的字段为汇总数据列。

以order by 的列为准，作为维度列。

其他的列保留第一行。

不在一个分区的数据不会被聚合。

设计聚合表的话，唯一键值、流水号可以去掉，所有字段全部是维度、度量或者时间戳。

能不能直接 select total\_amount from province\_name=’’ and create\_date=’xxx’ 来得到汇总值？

不行，可能会包含一些还没来得及聚合的临时明细

select sum(total\_amount) from province\_name=’’ and create\_date=’xxx’

第六章 SQL操作

基本上来说传统关系型数据库（以MySQL为例）的SQL语句，基本支持但是也有不一样的地方。这里不会从头讲解SQL语法只介绍Clickhouse与标准SQL（MySQL）不一致的地方。

## 6.1 Insert

基本与标准SQL（MySQL）基本一致

包括标准 insert into [table\_name] values(…),(….)

以及 从表到表的插入

insert into [table\_name] select a,b,c from [table\_name\_2]

## 6.2 Update 和 Delete

ClickHouse提供了Delete 和Update的能力，这类操作被称为Mutation查询，它可以看做Alter 的一种。

虽然可以实现修改和删除，但是和一般的OLTP数据库不一样，**Mutation语句是一种很“重”的操作，而且不支持事务**。

“重”的原因主要是每次修改或者删除都会导致放弃目标数据的原有分区，重建新分区。所以尽量做批量的变更，不要进行频繁小数据的操作。

删除操作

alter table t\_order\_smt delete where sku\_id ='sku\_001';

修改操作

alter table t\_order\_smt

update total\_amount=toDecimal32(2000.00,2)

where id =102;

由于操作比较“重”，所以 Mutation语句分两步执行，同步执行的部分其实只是进行新增数据新增分区和并把旧分区打上逻辑上的失效标记。知道触发分区合并的时候，才会删除旧数据释放磁盘空间。

## 6.3 查询操作

clickhouse基本上与标准SQL 差别不大。

支持子查询

支持CTE(with 子句)

支持各种JOIN， 但是JOIN操作无法使用缓存，所以即使是两次相同的JOIN语句，Clickhouse也会视为两条新SQL。

不支持窗口函数。

不支持自定义函数。

GROUP BY 操作增加了 with rollup\with cube\with total 用来计算小计和总计。

模拟数据

insert into t\_order\_mt

values(101,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

(102,'sku\_002',2000.00,'2020-06-01 12:00:00'),

(103,'sku\_004',2500.00,'2020-06-01 12:00:00'),

(104,'sku\_002',2000.00,'2020-06-01 12:00:00')

(105,'sku\_003',600.00,'2020-06-02 12:00:00'),

(106,'sku\_001',1000.00,'2020-06-04 12:00:00'),

(107,'sku\_002',2000.00,'2020-06-04 12:00:00'),

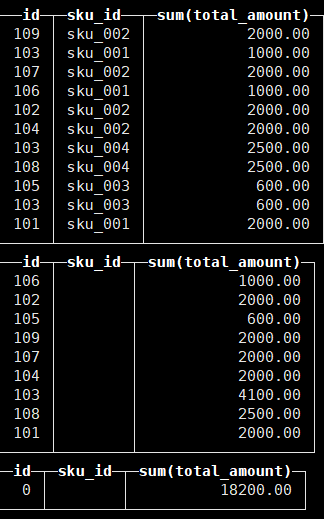
(108,'sku\_004',2500.00,'2020-06-04 12:00:00'),

(109,'sku\_002',2000.00,'2020-06-04 12:00:00'),

(110,'sku\_003',600.00,'2020-06-01 12:00:00')

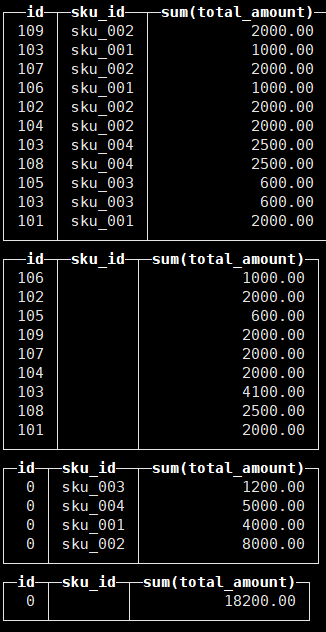
select id , sku\_id,sum(total\_amount) from t\_order\_mt group by id,sku\_id with rollup;

with rollup : 从右至左去掉维度进行小计。



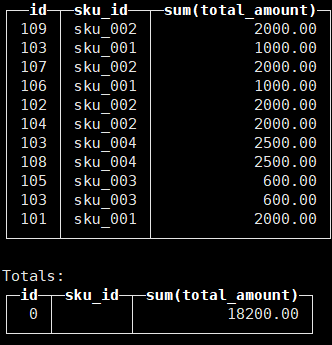
select id , sku\_id,sum(total\_amount) from t\_order\_mt group by id,sku\_id with cube;

with cube : 从右至左去掉维度进行小计，再从左至右去掉维度进行小计。



select id , sku\_id,sum(total\_amount) from t\_order\_mt group by id,sku\_id with totals;

with totals: 只计算合计。



## 6.4 alter操作

同mysql的修改字段基本一致，

**新增字段**

alter table **tableName** add column **newcolname** **String** after **col1**

**修改字段类型**

alter table **tableName** modify column **newcolname** **String** ；

**删除字段**

alter table **tableName** drop column **newcolname**  **;**

## 6.5 导出数据

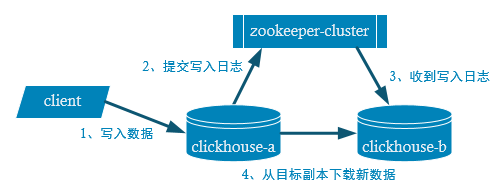
**clickhouse-client --query**  "select toHour(create\_time) hr ,count(\*) from test1.order\_wide where dt='2020-06-23' group by hr" **--format CSVWithNames**> ~/rs1.csv

支持格式的地址 <https://clickhouse.tech/docs/v19.14/en/interfaces/formats/#csvwithnames>

# 第七章 副本

副本的目的主要是保障数据的高可用性，即使一台clickhouse节点宕机，那么也可以从其他服务器获得相同的数据。

## 7.1 副本写入流程



## 7.2 配置

1. 这时需要启动zookeeper集群 和另外一台clickhouse 服务器

另外一台clickhouse服务器的安装完全和第一台一直即可。

3 在两台服务器的/etc/clickhouse-server/config.d目录下创建一个名为metrika.xml的配置文件：

<?xml version="1.0"?>

<yandex>

<zookeeper-servers>

<node index="1">

<host>hdp1</host>

<port>2181</port>

</node>

<node index="2">

<host>hdp2</host>

<port>2181</port>

</node>

<node index="3">

<host>hdp3</host>

<port>2181</port>

</node>

</zookeeper-servers>

</yandex>

在 /etc/clickhouse-server/config.xml 中增加

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



在两台电脑上分别建表

A机器

create table rep\_t\_order\_mt\_0105 (

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime

) engine =ReplicatedMergeTree('/clickhouse/tables/01/rep\_t\_order\_mt\_0105','rep\_hdp1')

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id,sku\_id);

B机器

create table rep\_t\_order\_mt\_0105 (

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime

) engine =ReplicatedMergeTree('/clickhouse/tables/01/rep\_t\_order\_mt\_0105','rep\_hdp2')

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id,sku\_id);

## 7.3 参数解释：

ReplicatedMergeTree 中，

第一参数是分片的zk\_path，一般按照：

/clickhouse/table/{shard}/{table\_name} 的格式写，如果只有一个分片就写01即可。

第二个参数是副本名称，相同的分片副本名称不能相同。

insert语句

insert into rep\_t\_order\_mt\_0105

values(101,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

(102,'sku\_002',2000.00,'2020-06-01 12:00:00'),

(103,'sku\_004',2500.00,'2020-06-01 12:00:00'),

(104,'sku\_002',2000.00,'2020-06-01 12:00:00')

(105,'sku\_003',600.00,'2020-06-02 12:00:00')

# 第八章 分片集群

副本虽然能够提高数据的可用性，降低丢失风险，但是对数据的横向扩容没有解决。每台机子实际上必须容纳全量数据。

要解决数据水平切分的问题，需要引入分片的概念。通过分片把一份完整的数据进行切分，不同的分片分布到不同的节点上。在通过Distributed表引擎把数据拼接起来一同使用。

Distributed表引擎本身不存储数据，有点类似于MyCat之于MySql，成为一种中间件，通过分布式逻辑表来写入、分发、路由来操作多台节点不同分片的分布式数据。

## 8.1 如何配置

配置的位置还是在之前的metrika.xml，配置分片如下的结构

<yandex>

<clickhouse\_remote\_servers>

<gmall\_cluster> <!-- 集群名称-->

<shard> <!--集群的第一个分片-->

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

<replica> <!—该分片的第一个副本-->

<host>hdp1</host>

<port>9000</port>

</replica>

<replica> <!—该分片的第二个副本-->

<host>hdp2</host>

<port>9000</port>

</replica>

</shard>

<shard> <!--集群的第二个分片-->

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

<replica> <!—该分片的第一个副本-->

<host>hdp3</host>

<port>9000</port>

</replica>

<replica> <!—该分片的第二个副本-->

<host>hdp4</host>

<port>9000</port>

</replica>

</shard>

<shard> <!--集群的第三个分片-->

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

<replica> <!—该分片的第一个副本-->

<host>hdp5</host>

<port>9000</port>

</replica>

<replica> <!—该分片的第二个副本-->

<host>hdp6</host>

<port>9000</port>

</replica>

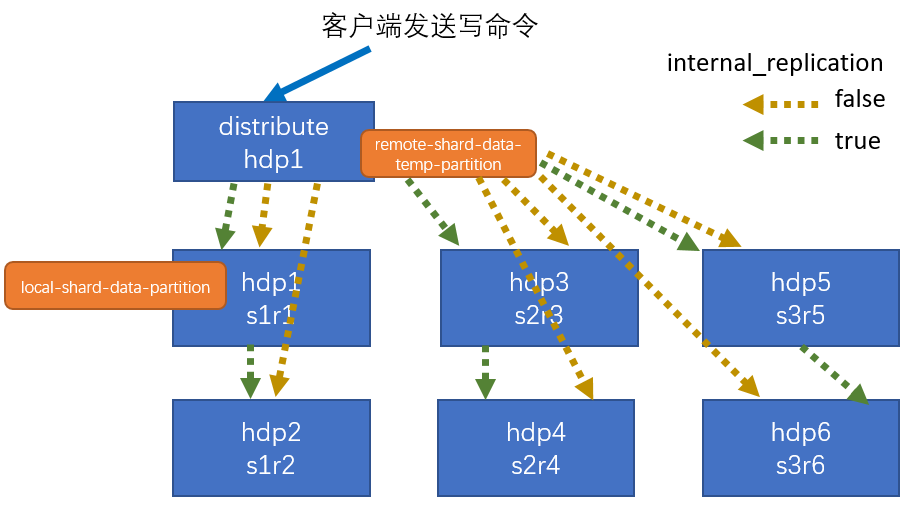
</shard>

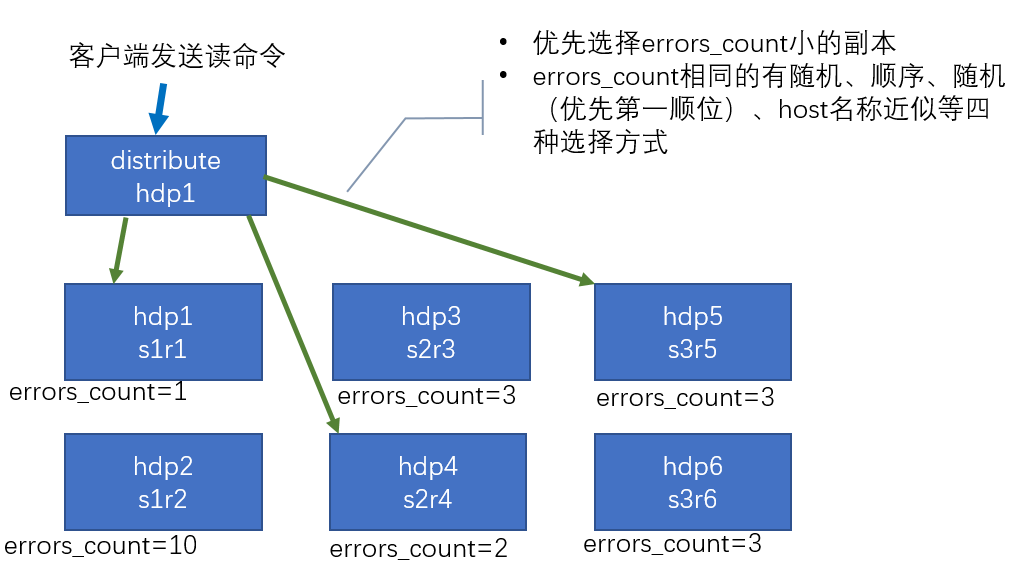
</gmall\_cluster>

</clickhouse\_remote\_servers>

</yandex>

## 8.2 读写原理：





## 8.3 三节点版本配置

metrika.xml

<yandex>

<clickhouse\_remote\_servers>

<gmall\_cluster> <!-- 集群名称-->

<shard> <!--集群的第一个分片-->

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

<replica> <!—该分片的第一个副本-->

<host>hdp1</host>

<port>9000</port>

</replica>

<replica> <!—该分片的第二个副本-->

<host>hdp2</host>

<port>9000</port>

</replica>

</shard>

<shard> <!--集群的第二个分片-->

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

<replica> <!—该分片的第一个副本-->

<host>hdp3</host>

<port>9000</port>

</replica>

</shard>

</gmall\_cluster>

</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>

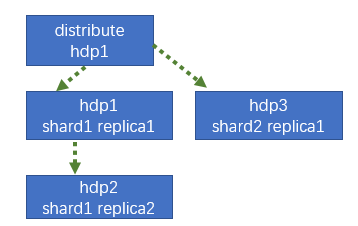
<macros>

<shard>**01**</shard> <!—不同机器放的分片数不一样-->

<replica>**rep\_1\_1**</replica> <!—不同机器放的副本数不一样-->

</macros>

</yandex>



|  |  |  |
| --- | --- | --- |
| hdp1 | hdp2 | hdp3 |
| <macros>  <shard>01</shard>  <replica>rep\_1\_1</replica>  </macros> | <macros>  <shard>01</shard>  <replica>rep\_1\_2</replica>  </macros> | <macros>  <shard>02</shard>  <replica>rep\_2\_1</replica>  </macros> |

create table st\_order\_mt\_0105 on cluster gmall\_cluster (

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime

) engine =ReplicatedMergeTree('/clickhouse/tables/{shard}/st\_order\_mt\_0105','{replica}')

partition by toYYYYMMDD(create\_time)

primary key (id)

order by (id,sku\_id);

## 8.4 Distribute 分布式表

create table st\_order\_mt\_0105\_all on cluster gmall\_cluster

(

id UInt32,

sku\_id String,

total\_amount Decimal(16,2),

create\_time Datetime

)engine = Distributed(gmall\_cluster,test0105, st\_order\_mt\_0105,hiveHash(sku\_id))

其中参数：

Distributed( 集群名称，库名，本地表名，分片键)

分片键必须是整型数字

也可以rand()

插入数据

insert into st\_order\_mt\_0105\_all

values(201,'sku\_001',1000.00,'2020-06-01 12:00:00') ,

(202,'sku\_002',2000.00,'2020-06-01 12:00:00'),

(203,'sku\_004',2500.00,'2020-06-01 12:00:00'),

(204,'sku\_002',2000.00,'2020-06-01 12:00:00')

(205,'sku\_003',600.00,'2020-06-02 12:00:00')

通过查询分布式表语句

SELECT \* FROM st\_order\_mt\_all

和 本地表

select \* from st\_order\_mt;

来观察数据的分布是否正确。

|  |  |
| --- | --- |
| st\_order\_mt\_all |  |
| hdp1:  st\_order\_mt |  |
| hdp2:  st\_order\_mt |  |
| hdp3:  st\_order\_mt |  |