

最强OLAP分析引擎-Clickhouse

快速精通一

==楼兰==

一、Clickhouse简介

- 1、什么是Clickhouse
- 2、Clickhouse适用场景。

二、Clickhouse环境安装

- 1、线上快速体验
- 2、本地快速部署
- 3、远程连接clickhouse
 - 3.1 打开远程连接控制
 - 3.2 其他方式访问clickhouse

三、Clickhouse使用篇

3.1、建库

- 3.1.1 Atomic 库引擎
- 3.1.2 MySQL库引擎

3.2、建表

- 3.2.1 数据类型
- 3.2.2 MergeTree 表引擎
 - partition by 分区键
 - order by 排序键
 - primary key 主键
 - TTL 数据存活时间
 - SAMPLE BY 数据抽样
- 3.2.3 ReplacingMergeTree
- MergeTree引擎族总结

3.3 数据使用

- 3.3.1、数据导入导出
- 3.3.2、数据修改
- 3.3.3、数据查询

一、Clickhouse简介

1、什么是Clickhouse

Clickhouse是由俄罗斯yandex公司开源的一个用于联机分析**OLAP**的**列式数据库**管理系统。他是使用**C++语言**编写的，支持SQL实时查询的**大型数据**管理系统。由于Clickhouse在大型数据集查询处理的高效表现，从2016年开源以来，就吸引了全球的目光，甚至一度登上github的关注度头把交易。

这一段介绍中标出了Clickhouse的几个显著特点。

- OLAP

Clickhouse的设计定位就是用于OLAP离线数据处理。相比于OLTP在线事务处理，Clickhouse更关注于对海量数据的计算分析，关注的是数据吞吐、查询速度、计算性能等指标。而对于数据频繁的修改变更，则不太擅长。所以Clickhouse通常用来构建后端的实时数仓或者离线数仓。

- 列式存储

Clickhouse是一个真正意义上的列式存储数据库。传统数据库存储数据都是按照数据行进行存储。

| Row | WatchID | JavaEnable | Title | GoodEvent | EventTime |
|-----|-------------|------------|--------------------|-----------|---------------------|
| #0 | 89354350662 | 1 | Investor Relations | 1 | 2016-05-18 05:19:20 |
| #1 | 90329509958 | 0 | Contact us | 1 | 2016-05-18 08:10:20 |
| #2 | 89953706054 | 1 | Mission | 1 | 2016-05-18 07:38:00 |
| #N | ... | ... | ... | ... | ... |

比如常用的MySQL，他的B+树的叶子节点就会整体保留一行数据。

这样的好处是，当想要查询某一条数据时，可以通过一次磁盘查找加顺序读取获得这一条完整的数据。

而Clickhouse存储数据的方式则是按照列来存储，将来自不同列的数据进行单独存储。实际上后面会介绍到，Clickhouse存储一个表数据时，就是以一列为一个文件进行存储的。

| Row: | #0 | #1 | #2 | #N |
|-------------|---------------------|---------------------|---------------------|-----|
| WatchID: | 89354350662 | 90329509958 | 89953706054 | ... |
| JavaEnable: | 1 | 0 | 1 | ... |
| Title: | Investor Relations | Contact us | Mission | ... |
| GoodEvent: | 1 | 1 | 1 | ... |
| EventTime: | 2016-05-18 05:19:20 | 2016-05-18 08:10:20 | 2016-05-18 07:38:00 | ... |

列式存储的数据库产品其实也有很多，像Druid数据库，InfiniDB等等很多。只是在国内其实用的还是比较少的。列式存储相比于行式存储在很多数据计算方面会体现出很多优势。例如通常一个计算过程都只会用到少数几个列的数据，这时行式存储就需要读取到相关行的所有列数据再进行过滤。而列式存储就可以直接读取这几个列的相关数据，而不用查找其他不关心的数据。

2、Clickhouse适用场景。

一个典型的OLAP场景主要是对海量数据进行更新，相比于我们常用的mysql等OLTP数据库，有一些很明显的特征。

- 绝大多数请求都是读请求。对数据的修改比较少或者几乎没有。
- 数据量很大。这个量即包括数据的行数，也包括数据的列数。也就是通常说的宽表。大部分情况下，对分布式表结构的要求是必须的。
- 数据通常以大的批次进行整体更新，而不是单行更新。这需要有很高的数据吞吐量。
- 对事务的要求不是必须的。对于数据一致性的要求不会太高。通常只要求数据最终一致性。

Clickhouse的数据吞吐量相当大，能够存储海量的数据，并能够以水平扩展的方式进行扩容。对大表的查询计算处理效率也非常高，甚至很多场景下都可以拥有媲美于关系型数据库的查询效率。官网给出的一些测试数据也大都都是 上千万行*数百列的数据规模。很多大规模的数据查询也都能轻松达到毫秒级别。

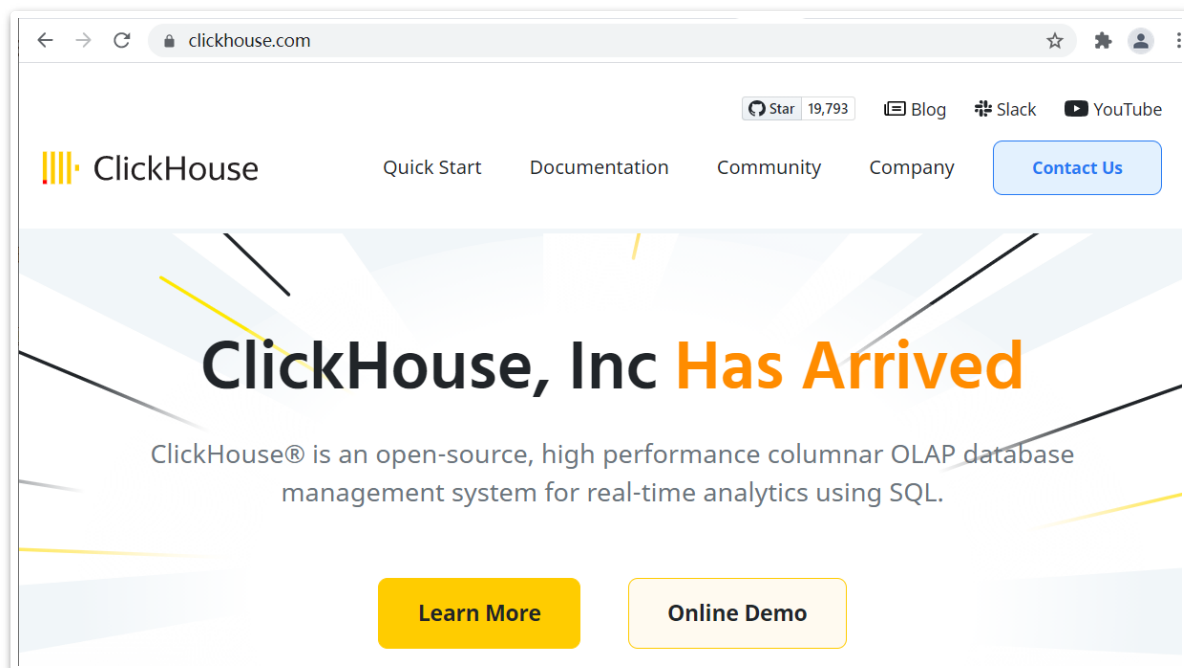
但是需要指出，Clickhouse高效性能的背后，肯定伴随计算机资源的大量消耗。Clickhouse对内存和CPU的占用率都非常高，一个很普通的查询都可能需要消耗非常多的资源。因此，Clickhouse的查询频率也不宜太高。过于频繁的连接或者并发查询甚至很容易导致服务直接崩溃。

综合Clickhouse的特点，他就非常适合用于后端数仓的建设。当然，这本身也是Clickhouse的设计目标。

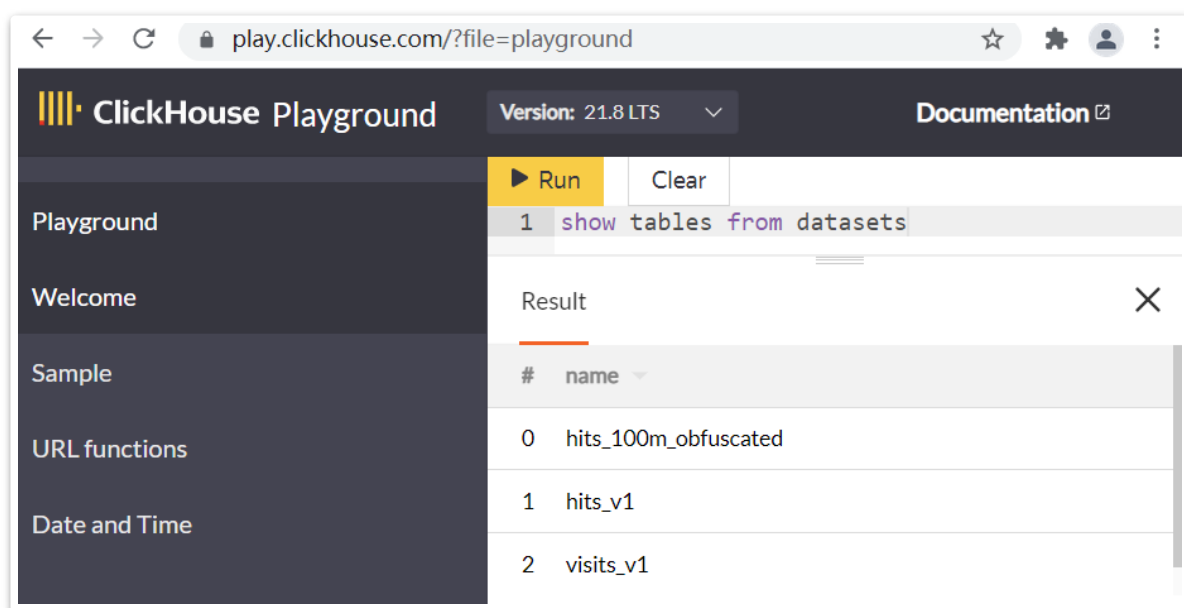
二、Clickhouse环境安装

1、线上快速体验

Clickhouse的官方网站是<https://clickhouse.com/>。他的LOGO是一个列式存储的示意图。



点击Online Demo按钮，可以直接访问线上的演示环境，直接运行Clickhouse。



线上环境中datasets库里有默认测试数据，这几个表都是千万级别行*上百列级别的大型测试数据，在上面可以直接执行SQL语句。并且在左侧的几个不同菜单中还包含默认的执行脚本。

你可以简单体验一下，执行`select count(WatchID) from hits_v1` 这样的统计SQL非常快，几乎都是毫秒级。但是执行`select * from hits_v1 limit 100`就慢很多。

2、本地快速部署

Clickhouse的官方文档非常详细，是学习Clickhouse最重要的资料。其中关于安装部署的章节地址是：<https://clickhouse.com/docs/zh/getting-started/install/>。

演示环境还是用三台CentOS服务器hadoop01,hadoop02,hadoop03三台机器。安装前，建议关闭三台服务器的防火墙以及SELinux安全组件，并打开操作系统的文件限制。

接下来这一章节将演示单机环境下的clickhouse安装。clickhouse的单机环境已经能够体现出非常强的性能。从单机扩展到集群是相当简单的，在后续clickhouse的备份以及分布式表章节用到集群时再一起分享。

clickhouse不需要依赖其他的组件，自己就能够提供非常强悍的数据处理性能。支持的安装环境非常多，安装方式也有很多种。这里，我们采用最为直观的tgz压缩包的方式进行安装。当然，如果是生产环境，建议的方式还是通过源码打包编译。

首先需要下载Clickhouse。下载地址：<https://repo.clickhouse.com/tgz/stable/>。在这个仓库中直接包含了Clickhouse的所有发布版本，俄罗斯简单直接的风格一览无余。从这里也能看出，Clickhouse的版本发布非常频繁。这里我们选取当前最新的21.9.4.35版本。这里面总共有四种tgz包是我们需要下载的，clickhouse-server-21.9.4.35.tgz, clickhouse-common-static-dbg-21.9.4.35.tgz, clickhouse-common-static-21.9.4.35.tgz, clickhouse-client-21.9.4.35.tgz。

然后将这四个压缩包解压并依次进行安装。

```
1 tar -xzvf clickhouse-common-static-$LATEST_VERSION.tgz
2 sudo clickhouse-common-static-$LATEST_VERSION/install/doinst.sh
3
4 tar -xzvf clickhouse-common-static-dbg-$LATEST_VERSION.tgz
5 sudo clickhouse-common-static-dbg-$LATEST_VERSION/install/doinst.sh
6
7 tar -xzvf clickhouse-server-$LATEST_VERSION.tgz
8 sudo clickhouse-server-$LATEST_VERSION/install/doinst.sh
9
10 tar -xzvf clickhouse-client-$LATEST_VERSION.tgz
11 sudo clickhouse-client-$LATEST_VERSION/install/doinst.sh
```

安装过程中，第1、第2、第4三个包的安装都没有输出，因为这三个包的安装过程只是简单的复制文件。安装第3个包clickhouse-server-21.9.4.35时，clickhouse会在数据库中创建一个默认的用户default，安装过程中，会需要给这个default用户输入一个密码。这里进行测试，就直接回车，不设置密码即可。正常情况下，安装完成会得到这样的输出日志。

```
1
2 Enter password for default user:
3 Password for default user is empty string. See /etc/clickhouse-
  server/users.xml and /etc/clickhouse-server/users.d to change it.
4 Setting capabilities for clickhouse binary. This is optional.
5
6 ClickHouse has been successfully installed.
7
8 Start clickhouse-server with:
9   sudo clickhouse start
10
11 Start clickhouse-client with:
12   clickhouse-client
```

安装完成后，就可以使用clickhouse start指令启动clickhouse服务了。

服务端还有其他一些指令，可以使用clickhouse --help查看

然后使用clickhouse-client就可以打开客户端。这样就可以使用SQL进行查询了。例如 show databases;查看已有的数据库。show tables from system; 查看系统表。

客户端启动时可以设置非常多的参数，可以使用clickhouse-client --help查看。常用的clickhouse -m表示支持多行SQL查询。

关于系统表的详细介绍可以参看官网文档 操作->系统表 章节。 <https://clickhouse.com/docs/zh/operations/system-tables/>

安装完成后，有几个重要的目录要记住：

- 执行脚本： /usr/bin/ 之前用到的clickhouse和clickhouse-client这些指令就被默认安装在这个目录。
- 配置文件： /etc/clickhouse-server/ 这个目录下的config.xml和users.xml是最为重要的两个配置文件。后续章节会介绍其中的重要配置。
- 运行日志： /var/log/clickhouse-server/ 服务运行的详细日志。
- 数据目录： /var/lib/clickhouse/ 这个目录包含了clickhouse运行时的所有数据文件。例如metadata目录下存放了所有表的元数据，可以看到，clickhouse就是以sql文件的方式保存表结构，启动时加载这些sql文件就完成了数据加载。而data目录下存放了所有的表数据。像之前看到的default和system两个默认的数据库就对应data目录下的两个文件夹。

另外，clickhouse在安装时，会默认创建一个clickhouse用户来部署这些文件。所以，如果不是使用root用户进行操作的话，需要注意下用户权限的问题。

3、远程连接clickhouse

3.1 打开远程连接控制

默认情况下，clickhouse服务只能在本地进行连接，远程机器是无法连接的。这点跟mysql是很类似的。因此，还需要做一些修改，让clickhouse可以远程访问。

配置方式直接修改clickhouse的config.xml配置文件。所在目录 /etc/clickhouse-server。将下面这一行注释打开。文件156行。

```
1 <listen_host>:::</listen_host>
```

然后重启clickhouse即可。

```
1 clickhouse restart
```

注意，重启时不能有客户端连接上。否则无法正常重启。

重启完成后，其他机器上就可以使用clickhouse-client命令行工具远程连接clickhouse服务了。只不过需要通过-h参数指定服务端机器名即可。

关于clickhouse-client命令行工具的其他使用方式，可以使用
clickhouse-client --help方式查看帮助。也可以查看官方文档：[https://
clickhouse.com/docs/zh/interfaces/cli/](https://clickhouse.com/docs/zh/interfaces/cli/)

3.2 其他方式访问clickhouse

clickhouse除了命令行客户端外，还提供了非常丰富的接入客户端。例如使用浏览器直接访问地址 `http://hadoop01:8123/?query=show databases` 就可以访问clickhouse的http客户端。这个8123端口可以在clickhouse的配置文件中定制。

同样在8123端口，clickhouse还提供了JDBC驱动程序来连接。目前官网提供了一个官方的驱动包以及两个第三方的驱动包。其中，官方JDBC驱动包的maven坐标是

```
1 <dependency>
2   <groupId>ru.yandex.clickhouse</groupId>
3   <artifactId>clickhouse-jdbc</artifactId>
4   <version>0.3.2</version>
5 </dependency>
```

引入这个驱动包后，就可以像连接其他关系型数据库一样访问clickhouse了。

```
1 Class.forName("ru.yandex.clickhouse.ClickHouseDriver");
2 Connection connection =
3   DriverManager.getConnection("jdbc:clickhouse://hadoop01:8123/default");
4 .....
5
```

当然，这个客户端工具也提供了自己封装的客户端，简化数据库访问。具体参见官方github仓库：<https://github.com/ClickHouse/clickhouse-jdbc>

另外还有两个第三方开发的JDBC驱动包。分别是 ClickHouse-Native-JDBC(仓库地址：<https://github.com/housepower/ClickHouse-Native-JDBC>) 和 clickhouse4j(仓库地址：<https://github.com/blynkkk/clickhouse4j>)

如果你觉得自己下载JDBC驱动包比较麻烦，那还有更简单的方式。clickhouse完全兼容最常用的mysql和postgresql两个数据库，可以用他们对应的JDBC驱动直接连接。只需要注意 mysql服务的默认端口是9004。Postgresql的默认端口是9005。

接下来，也可以将对应的jar包导入到一些第三方的客户端工具中，例如navicat, DataGrip等工具，像访问MySQL一样访问clickhouse。

从这个安装过程中可以体会到，虽然clickhouse底层的设计非常精妙，但是表现出来的实现方式却是非常简单直接。大部分的功能都是以一种统一有序的方式直接进行堆叠。

三、Clickhouse使用篇

clickhouse本身作为一个数据库，对普通增删改查的操作都是支持的。但是，他针对数仓的使用场景，又有非常多的高级特性。对这些高级特性的掌握程度将直接影响clickhouse的使用效率。实现一个同样的查询逻辑，不同的SQL写法在clickhouse上很容易体现出非常非常大的执行时长差别。所以在使用clickhouse时，需要对这些特性非常重视。

3.1、建库

使用数据库首先要建库，clickhouse提供了多种库引擎实现不同场景下的库声明。

3.1.1 Atomic 库引擎

这是clickhouse默认的库引擎。默认创建的default库就是使用的这种引擎。可以在建库时进行声明。

```
1 CREATE DATABASE test[ ENGINE = Atomic];
```

Atomic类型的数据库完全由clickhouse自己管理数据。每个数据库对应/var/lib/data/目录下的一个子目录。数据库中的每个表会分配一个唯一的UUID，数据存储在目录 /var/lib/clickhouse/store/xxx/xxxyyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyyyy/，其中xxxyyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyyyy是该表的UUID。

3.1.2 MySQL库引擎

clickhouse作为一个数据仓库，还提供了非常多与其他数据库整合的库引擎。最为常见的就是MySQL。

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，以方便您在ClickHouse与MySQL之间进行数据交换。MySQL数据库引擎会将其查询转换为MySQL语法并发送到MySQL服务器中，因此您可以执行诸如SHOW TABLES或SHOW CREATE TABLE之类的操作。

通过MySQL引擎可以省掉很多ETL的过程。例如下面的语句就可以在clickhouse中创建一个mysqldb。

```
1 CREATE DATABASE IF NOT EXISTS mysqldb ENGINE = MySQL('hadoop01:3306',  
  'testdb', 'root', 'root');
```

对于mysqldb库的操作，会转义成mysql语法，发送到相对应的MySQL中执行。

```
1 //使用mysqldb  
2 use mysqldb;  
3 //列出库中所有的表 --clickhouse中并没有建表  
4 show tables;  
5 //查询mysql中的数据  
6 hadoop01 :) select * from user;
```

接下来，可以像操作clickhouse自己的表一样进行insert\delete等操作。但是不能进行 RENAME、CREATE TABLE、ALTER操作。

这种库引擎，clickhouse本身并不存储数据，只是将请求转发到mysql。同样，clickhouse还提供了针对PostgreSQL、SQLite的库引擎。

是不是觉得只是请求转发还不够爽？性能不够高？clickhouse还提供了自己存储数据的物化引擎，针对MySQL的MaterializedMySQL引擎和针对PostgreSQL的MaterializedPostgreSQL引擎。这两个引擎都会将clickhouse服务器作为对应数据库的从库工作。通过执行日志实时将主库中的数据同步到clickhouse中。但是目前这两个引擎还在实验阶段。可以尝试，但不建议在生产上使用。

具体使用方式详见官方文档：<https://clickhouse.com/docs/zh/engine-s/database-engines/materialized-mysql/>

实际上，大部分场景下，我们就使用clickhouse自己的默认引擎就够了。而其他的引擎会通过定制的ETL过程来实现。但是clickhouse功能的朴实无华已经尽显无疑。

3.2、建表

3.2.1 数据类型

clickhouse的极简设计在基础数据类型中体现得尤为明显。

1、整型

clickhouse中的整型不像其他数据库中区分int, short, long等等这些类型，而是统一表示固定长度的整数，包括有符号整型和无符号整型。统一定义为Int，后面带上数字表示占用的字节数。

整型范围

- Int8-[-128:127] 占用8个字节，对应java中的byte
- Int16-[-32768:32767] 占用16个字节，对应java中的short
- Int32-[-2147483648:2147483647] 占用32个字节，对应java中的int
- Int64-[-9223372036854775808:9223372036854775807] 占用64个字节，对应java中的long

无符号整型范围

- UInt8-[0:255]
- UInt16-[0:65535]
- UInt32-[0:4294967295]
- UInt64-[0:18446744073709551615]

2、boolean布尔型

clickhouse中没有定义表示true和false的布尔类型数据，通常都是直接使用UInt8

3、浮点型

- Float32 - float
- Float64 - double

官方建议尽量使用整型来存储数据，将固定精度的数字转换成为整数值。例如时间用毫秒为单位保存。这是因为使用浮点型有精度丢失问题。例如执行 `select 1-0.9` 得到的结果将是 0.09999999999999998 而不是0.1。

浮点型一般用于数据值比较小，不设计大量的统计计算，精度要求也不高的场景。例如保存商品的重量。但是对于精度要求比较高的金额，就极不建议使用浮点型。而应该用Decimal型。

4、Decimal型

有符号的浮点数，可以在加、减和乘法运算过程中保持精度。对于除法，最低有效数字将被抛弃(不进行四舍五入)。通常有三种声明：Decimal32(s)、Decimal64(s)、Decimal128(s)。后面的s表示小数点后的数字位数。前面的32, 64, 128表示浮点精度，决定可以有多少个十进制数字(包含小数位)，也就代表不同的取值范围。

数据在底层会采用与自身位宽相同的有符号整数存储。而现代CPU不支持128位的数字，因此Decimal128上的操作需要由软件来进行模拟。所以Decimal128的运算速度会明显慢于Decimal32\Decimal64。也就是说尽量少用Decimal128。

5、字符型

clickhouse的字符型数据使用String进行声明。这个字符串可以是任意长度的。他可以包含任意的字节集，包含空字节。因此，字符串类型可以代替其他数据库中的VARCHAR、BLOB、CLOB等类型。

clickhouse中没有编码的概念。字符串可以是任意的字节集，按他们原本的方式进行存储和输出。对于不同的编码文本，clickhouse会有不同处理字符串的函数。比如 length函数可以计算字符串包含的字节数组的长度，而lengthUTF8函数是假设字符串以UTF-8编码，计算的字符串包含的Unicode字符的长度。

还有一个固定长度的字符串类型FixedString(N)，这个N就是要声明的字节数。如果字符串包含的字节数不足N，将会对字符串末尾进行空字节填充。如果字符串包含的字节数大于N，将会抛出异常。可以用来保存一些例如手机号码、IP地址这一类等长的规范数据。在实际开发中使用比较少。

6、枚举类型

包含Enum8和Enum16两种类型。Enum保存'string'=integer的对应关系。在clickhouse中，尽管用户使用的是字符串常量，但所有罕有Enum数据类型的操作都是按照其包含整数的值来执行的。这在性能方便比使用String数据类型更有效。Enum后面的8和16也是对应的整数值integer的位宽。

例如：先创建一个带枚举类型列的表。

```

1 CREATE TABLE t_enum
2 (
3     x Enum8('hello' = 1, 'world' = 2)
4 )
5 ENGINE = TinyLog

```

这个x列只能存储类型定义中列出来的值，hello 或者是 world。尝试insert插入其他值时会抛异常。

```

1 :) INSERT INTO t_enum VALUES ('hello'), ('world'), ('hello')
2
3 INSERT INTO t_enum VALUES
4
5 Ok.
6
7 3 rows in set. Elapsed: 0.002 sec.
8
9 :) insert into t_enum values('a')
10
11 INSERT INTO t_enum VALUES
12
13 Exception on client:
14 Code: 49. DB::Exception: Unknown element 'a' for type Enum8('hello' = 1,
    'world' = 2)

```

从表中查询数据时，clickhouse会返回前面的字符串值。

```

1 SELECT * FROM t_enum
2
3 ┌x┐
4 │ hello │
5 │ world │
6 │ hello │
7 └┐

```

如果需要查看对应行的数值，则必须将Enum值转换为整数类型。

```

1 SELECT CAST(x, 'Int8') FROM t_enum
2
3 ┌CAST(x, 'Int8')┐
4 │                1 │
5 │                2 │
6 │                1 │
7 └┐

```

枚举类型在开发中可以很方便的代替字典表，优化一些例如状态、类型这样的字段。但是实际使用时，如果枚举值发生变化，就会带来非常多的维护成本，甚至会带来数据丢失的问题。因此，枚举类型要谨慎使用。

7、数组类型

类型声明： `array(T)` 。表示一个由T类型元素组成的数组。T可以是任意类型，甚至也可以是数组类型。但是不建议使用多位数组，clickhouse对多维数组的支持有限。例如在MergeTree引擎中就不能存储多维数组。

示例：

```
1 :) SELECT array(1, 2) AS x, toTypeName(x)
2
3 SELECT
4     [1, 2] AS x,
5     toTypeName(x)
6
7 ┌-x-----┐toTypeName(array(1, 2))┐
8 │ [1,2] │ Array(UInt8) │
9 └-----┘
10
11 1 rows in set. Elapsed: 0.002 sec.
12
13 :) SELECT [1, 2] AS x, toTypeName(x)
14
15 SELECT
16     [1, 2] AS x,
17     toTypeName(x)
18
19 ┌-x-----┐toTypeName([1, 2])┐
20 │ [1,2] │ Array(UInt8) │
21 └-----┘
22
23 1 rows in set. Elapsed: 0.002 sec.
```

8、时间类型

时间类型是每个数据库都要处理的类型。clickhouse的时间类型声明相对简单很多。在clickhouse中有三种时间类型

- `Date` 可以接受一个 **年-月-日** 格式的字符串。例如 `'2021-10-13'`。
- `Datetime` 可以接受一个 **年-月-日 时:分:秒** 格式的字符串。例如 `'2021-10-13 20:50:10'`。

- Datetime64 可以接受一个 **年-月-日 时:分:秒.毫秒** 格式的字符串。例如 '2021-10-13 20:50:10.232'。

9、可为空类型

绝大部分的基础类型都可以通过在前面添加一个Nullable()声明来允许接受Null空值。例如Nullable(Int8)类型的列可以存储Int8类型的值，没有值的行将存储NULL。

Nullable类型字段不能包含在表索引中。并且使用Nullable几乎总是对性能产生负面影响，在设计数据库时要尽量避免使用Nullable。例如对于字符串，可以用空字符串代替Null。而对于整型数据，可以用无业务意义的数字例如-1来表示Null

```
1 CREATE TABLE t_null(x Int8, y Nullable(Int8)) ENGINE TinyLog;
2 INSERT INTO t_null VALUES (1, NULL), (2, 3);
3 SELECT x + y FROM t_null;
4 ┌plus(x, y)┐
5 │      NULL │
6 │          5 │
7 └──────────┘
```

clickhouse中还设计了很多非常有特色的数据类型，例如Geo,Map,Tuple,UUID等类型。具体参见官方文档。<https://clickhouse.com/docs/zh/sql-reference/data-types/>

3.2.2 MergeTree 表引擎

就像MySQL提供了InnoDB和MyISAM等等多种数据引擎来对表进行管理一样，clickhouse也提供了非常多的引擎来对表进行管理，只是clickhouse的表引擎更多，功能更强大。表引擎也是clickhouse非常有特色的一个功能。表引擎决定了一个表的所有数据属性，包括

- 数据的存储方式和位置，写到哪里以及从哪里读取数据
- 支持哪些查询以及如何支持。
- 并发数据访问。
- 索引的使用（如果存在）。
- 是否可以执行多线程请求。
- 数据复制参数。

clickhouse中的表引擎非常丰富，有好几十中。整体可以分为四类。

- MergeTree 合并树家族：这是适用于高负载任务的最通用同时功能最强大的表引擎。这一类引擎的共同特点是可以快速插入数据并进行后续的后台数据处理。是clickhouse默认的也是最为重要的引擎。
- Log 日志系列：具有最小功能的轻量级引擎。用于快速写入许多小表(最多约100万行)，并在以后整体读取这些数据。例如常用的滚动日志。例如TinyLog引擎，以列文件的形式保存在磁盘上，不支持索引，没有并发控制，通常只用于练习。
- Integration Engines 集成引擎：用于与其他的数据存储与处理系统集成的引擎。通常可用来简化一些ETL的工作。例如同样有MySQL的表引擎，将对表的查询语句转发到远程MySQL数据库中。另外，可以看到，clickhouse支持的集成表引擎比库引擎丰富很多。
- Special Engines 特别引擎：用于其他特定功能的引擎。比如使用内存表、字典表等。

这其中，我们最关注的当然是MergeTree合并树家族，后续关于clickhouse表的分享也都基于MergeTree引擎。其他几种类型的表引擎可以在用到的时候去官网查一下即可。

官方资料查询地址：<https://clickhouse.com/docs/zh/engines/table-engines/>

clickhouse中最强大的表引擎当属MergeTree合并树引擎以及该系列*MergeTree中的其他引擎。其地位堪比MySQL中的innodb。

MergeTree系列的表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已存储的数据，这种策略会高效很多。他的主要特点：

- 存储的数据按主键排序。这样你能够创建一个小型的稀疏索引来加快数据检索。
- 如果指定了分区键的话，可以使用分区。查询中国指定分区键时，clickhouse会自动截取分区数据，能有效增加查询性能。
- 支持数据副本
- 支持数据采样。如果需要的话，可以给表设置一个采样方式。

基于MergeTree引擎的建表语句是这样：

```
1 CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
2 (
3     name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
4     name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
```

```

5      ...
6      INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
7      INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
8  ) ENGINE = MergeTree()
9  ORDER BY expr
10 [PARTITION BY expr]
11 [PRIMARY KEY expr]
12 [SAMPLE BY expr]
13 [TTL expr [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'], ...]
14 [SETTINGS name=value, ...]

```

主要通过Engine指定表引擎，然后下面指定一些相关的参数。

例如：

```

1  create table t_stock(
2      id UInt32,
3      sku_id String,
4      total_amount Decimal(16,2),
5      create_time Datetime
6  ) engine =MergeTree()
7      partition by toYYYYMMDD(create_time)
8      primary key (id)
9      order by (id,sku_id);

```

对于MergeTree，SETTINGS中的大部分蚕食都有了默认值，可以不用设置。重点关注的就是以下几个配置。partition by 分区键，primary key 主键 以及 order by 排序键。下面来逐一进行分享。

partition by 分区键

1> 分区键的作用

分区键的作用主要是降低数据扫描的范围，优化查询速度。例如示例中，按天进行了分区，当查询的where条件中指定了日期条件，就只需要去扫描对应日期的数据，而不用进行全表扫描了。使用分区后，涉及到跨分区的查询操作，clickhouse将会以分区为单位进行并行处理。在clickhouse中这是一个可选项，如果不填，相当于只用一个分区。

2> 分区表的数据目录

MergeTree引擎默认是以列文件+索引文件+表定义文件共同描述一个表。这些文件都在clickhouse的本地数据磁盘当中(默认/var/lib/clickhouse目录)。如果设定了分区，那么这些文件都会保存在不同的分区目录中。

例如我们在示例表中插入几条数据：

```
1 insert into t_stock values
2 (101, 'sku_002', 2000.00, '2020-06-01 11:00:00'),
3 (102, 'sku_004', 2500.00, '2020-06-01 12:00:00'),
4 (103, 'sku_002', 2000.00, '2020-06-02 13:00:00'),
5 (104, 'sku_002', 12000.00, '2020-06-03 13:00:00'),
6 (105, 'sku_002', 600.00, '2020-06-04 12:00:00');
```

然后进入clickhouse的数据目录来看看。

在default库对应的元数据目录 /var/lib/clickhouse/metadata/default 下，保存了t_stock.sql。这个就是clickhouse保存的元数据信息。直接就是一个简单粗暴的SQL语句。

```
1 ATTACH TABLE _ UUID 'a6f7175a-71e0-40e5-a6f7-175a71e040e5'
2 (
3     `id` UInt32,
4     `sku_id` String,
5     `total_amount` Decimal(16, 2),
6     `create_time` DateTime
7 )
8 ENGINE = MergeTree()
9 PARTITION BY toYYYYMMDD(create_time)
10 PRIMARY KEY id
11 ORDER BY (id, sku_id)
12 SETTINGS index_granularity = 8192
```

只不过对我们写的建表语句做了一点小小的修改。

然后进入t_stock表的数据目录， /var/lib/clickhouse/data/default/t_stock。这个目录实际上是指向store目录下的一个数据目录的软连接。

```
1 [root@hadoop01 default]# ll
2 总用量 0
3 lrwxrwxrwx 1 clickhouse clickhouse 67 10月 13 15:07 t_stock ->
  /var/lib/clickhouse/store/a6f/a6f7175a-71e0-40e5-a6f7-175a71e040e5/
```

store下的目录之前提过，就是表的UUID。

这个目录下就保存了t_stock表的数据。可以看到里面的目录结构：

```
1 [root@hadoop01 t_stock]# ll
2 总用量 4
3 drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200601_1_1_0
4 drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200602_2_2_0
5 drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200603_3_3_0
6 drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200604_4_4_0
7 drwxr-x--- 2 clickhouse clickhouse 6 10月 13 15:07 detached
8 -rw-r----- 1 clickhouse clickhouse 1 10月 13 15:07 format_version.txt
```

对于20200601_1_1_0这样的目录，用下划线拆分成了四个部分。20200601就表示对应的分区。这个分区下的数据就保存在这个目录下。而后面的1_1_0。前面两个1，表示这个目录包含的分区最小块编号和最大块编号，最后一个0表示数据合并次数。关于分区合并，会在下面讲到。

关于数据分区：如果分区键没有指定，会生成一个all分区。如果分区键是String,Float型，会经过hash产生一个hashid作为分区值。

这个目录下就记载了当前分区的详细信息。里面有几个关键的文件：

```
1 bin文件：数据文件
2 mrk文件：标记文件
3     标记文件在 idx索引文件 和 bin数据文件 之间起到了桥梁作用。
4     以mrk2结尾的文件，表示该表启用了自适应索引间隔。
5 primary.idx文件：主键索引文件，用于加快查询效率。
6 minmax_create_time.idx：分区键的最大最小值。
7 checksums.txt：校验文件，用于校验各个文件的正确性。存放各个文件的size以及hash值。
8 columns.txt：表的结构信息
9 count.txt：当前分区的数据条数。 --所以对于clickhouse来说，查表的行数非常非常快。
```

设置了分区键后，在客户端使用select * from t_stock就能看到分区的结果。

```
hadoop01 :) select * from t_stock;
```

```
SELECT *  
FROM t_stock
```

Query id: c6bc5290-aa60-4529-8848-a698d97b877c

| id | sku_id | total_amount | create_time |
|-----|---------|--------------|---------------------|
| 105 | sku_002 | 600 | 2020-06-04 12:00:00 |
| id | sku_id | total_amount | create_time |
| 104 | sku_002 | 12000 | 2020-06-03 13:00:00 |
| id | sku_id | total_amount | create_time |
| 103 | sku_002 | 2000 | 2020-06-02 13:00:00 |
| id | sku_id | total_amount | create_time |
| 101 | sku_002 | 2000 | 2020-06-01 11:00:00 |
| 102 | sku_004 | 2500 | 2020-06-01 12:00:00 |

5 rows in set. Elapsed: 0.022 sec.

使用第三方工具是看不到分区结果的。

3> 分区合并

MergeTree引擎底层使用一种类似于LSM树的结构来保存数据。任何一次对数据的修改都会临时产生一个分区，而不会修改已有的分区。写入后的某个时刻，clickhouse会在后台自动执行合并操作。这个这个间隔时间是未知的，大概在10~15分钟左右。如果等不及，也可以执行手动合并。合并指令

```
1 optimize table t_stock final;
```

下面。我们再次插入几条20200601这一个分区内的测试数据。

```
1 insert into t_stock values  
2 (101,'sku_002',2000.00,'2020-06-01 14:00:00'),  
3 (102,'sku_004',2500.00,'2020-06-01 15:00:00'),  
4 (103,'sku_002',2000.00,'2020-06-01 16:00:00'),  
5 (104,'sku_002',12000.00,'2020-06-01 17:00:00');
```

插入完成后，再次查询t_stock的数据，会成这样：

```
hadoop01 :) select * from t_stock;
```

```
SELECT *  
FROM t_stock
```

Query id: a51c513e-b5a3-4706-ab8c-11331437d690

| id | sku_id | total_amount | create_time |
|-----|---------|--------------|---------------------|
| 104 | sku_002 | 12000 | 2020-06-03 13:00:00 |
| 105 | sku_002 | 600 | 2020-06-04 12:00:00 |
| 101 | sku_002 | 2000 | 2020-06-01 14:00:00 |
| 102 | sku_004 | 2500 | 2020-06-01 15:00:00 |
| 103 | sku_002 | 2000 | 2020-06-01 16:00:00 |
| 104 | sku_002 | 12000 | 2020-06-01 17:00:00 |
| 101 | sku_002 | 2000 | 2020-06-01 11:00:00 |
| 102 | sku_004 | 2500 | 2020-06-01 12:00:00 |
| 103 | sku_002 | 2000 | 2020-06-02 13:00:00 |

9 rows in set. Elapsed: 0.038 sec.

从结果中可以看出 2020-06-01这个分区的数据没有合并。


然后我们再去t_stock表的数据目录下看一下clickhouse的数据目录。

```
[root@hadoop01 t_stock]# ll  
总用量 4  
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200601_1_1_0  
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:35 20200601_5_5_0  
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200602_2_2_0  
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200603_3_3_0  
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200604_4_4_0  
drwxr-x--- 2 clickhouse clickhouse 6 10月 13 15:07 detached  
-rw-r----- 1 clickhouse clickhouse 1 10月 13 15:07 format_version.txt
```

可以看到，新插入的数据进入了20200601_5_5_0这个目录。这表示新插入的数据进入了20200601分区的5号数据块。

这时，我们执行一次手动合并 `optimize table t_stock final;` 之后再来看下数据目录：

```
[root@hadoop01 t_stock]# ll
总用量 4
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200601_1_1_0
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:38 20200601_1_5_1
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:35 20200601_5_5_0
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200602_2_2_0
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:38 20200602_2_2_1
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200603_3_3_0
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:38 20200603_3_3_1
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:14 20200604_4_4_0
drwxr-x--- 2 clickhouse clickhouse 203 10月 13 15:38 20200604_4_4_1
drwxr-x--- 2 clickhouse clickhouse 6 10月 13 15:07 detached
-rw-r----- 1 clickhouse clickhouse 1 10月 13 15:07 format_version.txt
[root@hadoop01 t_stock]#
```



这个结果表明 原有的20200601_1_1_0和20200601_5_5_0 这两个数据块，被合并到了20200601_1_5_1这个数据块中。新生成的数据块包含了20200601分区下，从1号数据块到5号数据块的内容，合并次数为1。而后续对于t_stock表的查询，都会走这个新生成的数据目录来查。原有的两个数据目录会在未来clickhouse进行全局合并时删除。

order by 排序键

order by 排序键 指定分区内的数据按照哪些字段排序进行有序保存。这是MergeTree中唯一的一个必填项。

数据有序保存对于clickhouse底层的数据处理是相当重要的，在海量数据场景下，实现快速检索、去重、汇总等计算都离不开数据有序性的支持。这里需要注意的是，clickhouse的数据是分区内局部有序的。实际上这也比较好理解，因为clickhouse对于数据的处理就是以分区作为最小维度的。

分区键的设置对于主键也是有影响的。在clickhouse中，如果不设置表的主键，他就会以排序键来对数据进行检索等数据处理。这里要注意的是，如果设置主键，主键必须是order by的前缀字段。例如order by 排序键设置为(id,sku_id)，那么主键只能是 id 或者是 (id,sku_id)。

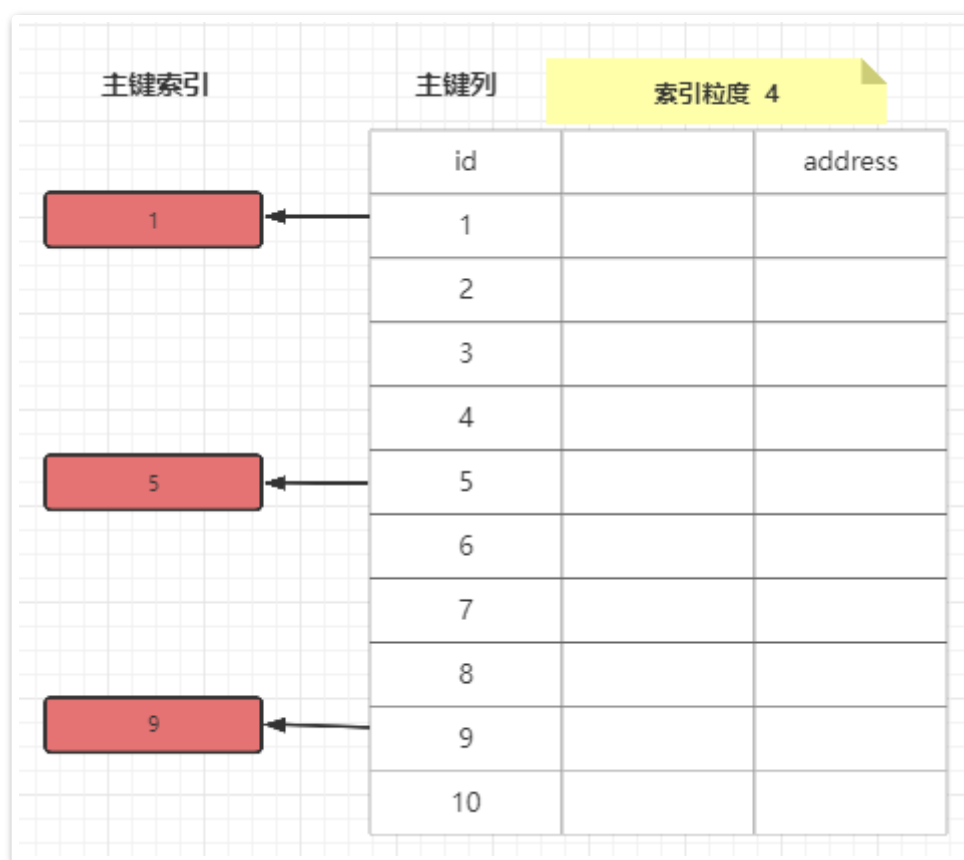
primary key 主键

主键的作用是为了加快数据检索的。clickhouse中的主键与其他数据库有点不太一样，他并不要求主键的数据具有唯一性。

我们之前已经看到，在clickhouse的metadata文件中保存的关于t_stock表的sql语句。而在那个sql文件当中，clickhouse在我们自定义的建表语句之后，加了一个默认的参数 index granularity，指定了值是8192。这是clickhouse中主键的一个重要作用。

index granularity，直接翻译的话是叫做索引粒度，是指在稀疏索引中两个相邻索引对应数据的间隔。clickhouse给出的默认值是8192。官方不建议修改这个值，但是有一种情况可能需要调整这个值，那就是数据中有非常大量的重复值，例如一个分区中几万行数据的主键列数值都是一样的，当然很明显，这种情况是非常少见的。

首先需要理解一个概念，稀疏索引。稀疏索引的概念非常类似于在Redis中经常提到的调表skiplist。也就是在构建索引数据时，并不记录每一个主键的数值。而是按照一定的稀疏度，记录几个节点的索引数据。而这些记录的数据，就保存在分区所在的数据目录中。



稀疏索引的好处是可以减少数据的检索次数。每次根据一个主键进行数据查找时，可以根据稀疏索引确定数据所在的范围，然后再在选定的范围内进行逐行扫描，就能快速定位到目标数据。

clickhouse中的主键相当于给主键列的数据建立了一级索引，而实际上，在一级索引的基础上，clickhouse还提供了二级索引的功能，相当于给一级索引再建立一个索引。二级索引的目的同样也是为了加快数据检索速度。例如

```

1 create table t_stock_2(
2     id UInt32,
3     sku_id String,
4     total_amount Decimal(16,2),
5     create_time Datetime,
6     INDEX secondIndex total_amount TYPE minmax GRANULARITY 5
7 ) engine =MergeTree
8     partition by toYYYYMMDD(create_time)
9     primary key (id)
10    order by (id, sku_id);

```

在total_amout列上，就设定了一个类型为minmax的二级索引(还有一些其他的索引类型，例如常见的bloomfliter。具体查看官网)，名字为secondIndex。建立二级索引时，还指定了一个GRANULARITY参数，翻译过来也是粒度的意思。这个粒度表示对一级索引进行聚合的粒度。这是什么意思呢？

例如按照上面的示例图，对于一级索引，按照GRANULARITY粒度为4，就会划分为[1,3],[3,6],[6,9],[9,12]...这样的一些区间。而二级索引按照3的粒度，就会将三个区间聚合到一起，形成[1,9],[9,18]这样的区间信息。当对数据进行检索时，就可以先按二级索引先确定一个初略的范围，再按照一级索引确定数据遍历的范围。

TTL 数据存活时间

TTL即Time To Live。可以用来指定行存储的持续时间。MergeTree可以针对表或者列声明数据存活时间。设置TTL需要指定一个表达式来表示数据的存活时间。表达式中必须存在至少一个表示时间的Date或DateTime类型的列。比如 TTL date + INTERVAL 1 DAY 。也就是说，存活时间必须跟数据相关联。

列级TTL

可以在列上直接声明TTL规则。例如下面的语句就可以声明total_amount字段的存活时间为create_time创建时间后的10秒钟。

```

1 CREATE TABLE example_table
2 (
3     d DateTime,
4     a Int TTL d + INTERVAL 1 MONTH,
5     b Int TTL d + INTERVAL 1 MONTH,
6     c String
7 )
8 ENGINE = MergeTree
9 PARTITION BY toYYYYMM(d)
10 ORDER BY d;

```

当列中的值过期时，clickhouse会将他们替换成该列数据类型的默认值。如果某个数据块中列的所有值都过期了，那么clickhouse会从文件系统中的数据块中直接删除这一列。

列式TTL不能用于主键。

表级TTL

设置表级TTL时，除了设置一个过期表达式之外，还可以配置一个数据移除规则。完整的声明指令是这样的：

```
1  TTL expr
2  [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'][, DELETE|TO DISK 'aaa'|TO VOLUME
   'bbb'] ...
3  [WHERE conditions]
4  [GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]] ]
```

但是通常不会用得这么复杂。一般用到后面第一个中括号的可选项就差不多了。这里是定义clickhouse如何移除过期数据。

- Delete - 删除过期的行 默认行为
- TO DISK 'aaa' - 将数据块移动到磁盘'aaa'
- TO VOLUME 'bbb' - 将数据块移动到卷 'bbb'
- GROUP BY - 聚合过期的行

后面的where 可以指定哪些过期的行为会被删除或聚合(不适用于数据移动)。

例如：

```
1  CREATE TABLE example_table
2  (
3      d DateTime,
4      a Int
5  )
6  ENGINE = MergeTree
7  PARTITION BY toYYYYMM(d)
8  ORDER BY d
9  TTL d + INTERVAL 1 MONTH [DELETE],
10     d + INTERVAL 1 WEEK TO VOLUME 'aaa',
11     d + INTERVAL 2 WEEK TO DISK 'bbb';
```

SAMPLE BY 数据抽样

数据抽样同样用于大数据分析，可以极大提升数据分析的性能。采样修饰符只能用在MergeTree的表中才有效，并且抽样表达式指定的列，必须包含在主键中。进行了采样声明后，就可以在查询时进行采样查询。

例如 官方提供的测试数据集hits_v1，在表声明时指定了采样规则

```
1 | SAMPLE BY intHash32 (UserID)
```

接下来就可以在查询时指定采样效率。

```
1 | SELECT Title, count(*) AS PageViews
2 | FROM hits_v1
3 | SAMPLE 0.1 #代表采样 10%的数据, 也可以是具体的条数
4 | WHERE CounterID =57
5 | GROUP BY Title
6 | ORDER BY PageViews DESC LIMIT 1000
```

这个采样查询是在满足条件的结果集中随机抽取10%的数据。

3.2.3 ReplacingMergeTree

下面我们介绍一个MergeTree家族中用得比较多的一个表引擎 ReplacingMergeTree。这个表引擎与MergeTree的不同之处在于他会删除排序值相同的重复项。这个去重的功能在实际开发中还是经常会要用到的。

但是要注意，ReplactingMergeTree的数据去重只会在数据合并期间进行。对应之前数据合并的示例就比较容易理解。正常情况下，数据合并是在后台一个不确定的时间进行，这个时间是无法预先规划的。当然如果确实需要，可以使用optimize语句手动发起合并，但是这显然是不建议的，因为optimize语句会引发数据的大量读写，会严重影响数据库的性能。

所以，ReplacingMergeTree适用于在后台清除重复的数据用来节省空间，但是他并不保证没有重复的数据出现。也就是说他只保证数据的最终一致，而不能保证强一致。

使用ReplacingMergeTree的建表指令如下：

```

1 CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
2 (
3     name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
4     name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
5     ...
6 ) ENGINE = ReplacingMergeTree([ver])
7 [PARTITION BY expr]
8 [ORDER BY expr]
9 [SAMPLE BY expr]
10 [SETTINGS name=value, ...]

```

基本上跟MergeTree是差不多的。最大的区别就在于ReplactingMergeTree需要指定一个参数ver。这个参数就表示版本列。类型必须是UInt*,Date或者DateTime。这是个可选的参数。

这个参数是用来指定数据去重的规则。可以想象，当数据集中出现了多条重复的数据，ReplacingMergeTree会在这一批重复数据中保存版本列的数据最大的那一条数据，而其他数据则标记为过期。如果没有指定版本列，则会默认保留最后插入的那一条数据。

例如可以做个实验

```

1 create table t_stock_merge(
2     id UInt32,
3     sku_id String,
4     total_amount Decimal(16,2) ,
5     create_time Datetime
6 ) engine =ReplacingMergeTree(create_time)
7 partition by toYYYYMMDD(create_time)
8 primary key (id)
9 order by (id, sku_id);

```

然后插入一批带有重复id和sku_id的数据

```

1 insert into t_stock_merge values
2 (101,'sku_001',1000.00,'2020-06-01 12:00:00') ,
3 (102,'sku_002',2000.00,'2020-06-01 11:00:00'),
4 (102,'sku_004',2500.00,'2020-06-01 12:00:00'),
5 (102,'sku_002',2000.00,'2020-06-01 13:00:00'),
6 (102,'sku_002',12000.00,'2020-06-01 13:00:00'),
7 (102,'sku_002',600.00,'2020-06-02 12:00:00');

```

然后查询t_stock_merge表中的数据：

```
hadoop01 :) insert into t_stock_merge 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');
```

```
INSERT INTO t_stock_merge VALUES
```

Query id: e67a98d4-8325-45f1-8e2a-d412ce138ac9

Ok.

6 rows in set. Elapsed: 0.016 sec.

```
hadoop01 :) select * from t_stock_merge;
```

```
SELECT *
FROM t_stock_merge
```

Query id: 7bf63996-7378-425b-9645-cbb9abda85dc

| id | sku_id | total_amount | create_time |
|-----|---------|--------------|---------------------|
| 102 | sku_002 | 600 | 2020-06-02 12:00:00 |
| 101 | sku_001 | 1000 | 2020-06-01 12:00:00 |
| 102 | sku_002 | 12000 | 2020-06-01 13:00:00 |
| 102 | sku_004 | 2500 | 2020-06-01 12:00:00 |

可以看到，id为102，sku为sku_002的一组数据已经完成了合并。

对于ReplacingMergeTree需要注意的几个重点：

- ReplacingMergeTree是按照order by 指定的排序键作为判断重复的标准。
- 他的去重只限定在一个分区中，不能跨区去重。
- 对于判断为重复的数据，保留版本字段最大的一条数据，如果没有指定版本值或者版本值也有重复的，就会保留最后插入的一条数据。
- ReplacingMergeTree并不能始终保证数据是完全去重的。数据去重只会发生在**同一批插入数据** 以及 **后台数据合并** 这两个时机。

同样，我们再来试验一下 SummingMergeTree。他在MergeTree的基础上，在进行数据合并时，clickhouse会把所有具有相同主键的行合并成一行。新合并的行包含了被合并的行中具有数值数据类型的列的汇总值。

例如在建表时，通过engine参数指定SummingMergeTree，SummingMergeTree需要指定参数，表明按照哪个列进行聚合分析。

```

1 create table t_stock_sum(
2     id UInt32,
3     sku_id String,
4     total_amount Decimal(16,2) ,
5     create_time Datetime
6 ) engine =SummingMergeTree(total_amount)
7   partition by toYYYYMMDD(create_time)
8   primary key (id)
9   order by (id,sku_id );

```

插入一些测试数据：

```

1 insert into t_stock_sum values
2 (101,'sku_001',1000.00,'2020-06-01 12:00:00') ,
3 (102,'sku_002',2000.00,'2020-06-01 11:00:00'),
4 (102,'sku_004',2500.00,'2020-06-01 12:00:00'),
5 (102,'sku_002',2000.00,'2020-06-01 13:00:00'),
6 (102,'sku_002',12000.00,'2020-06-01 13:00:00'),
7 (102,'sku_002',600.00,'2020-06-02 12:00:00');

```

接下来再次查询数据时，会将totalamount进行聚合。

```
hadoop01 :) select * from t_stock_sum
:-] ;
```

```
SELECT *
FROM t_stock_sum
```

Query id: 56b0a881-1301-4df6-8005-176d487df082

| id | sku_id | total_amount | create_time |
|-----|---------|--------------|---------------------|
| 102 | sku_002 | 600 | 2020-06-02 12:00:00 |
| id | sku_id | total_amount | create_time |
| 101 | sku_001 | 1000 | 2020-06-01 12:00:00 |
| 102 | sku_002 | 16000 | 2020-06-01 11:00:00 |
| 102 | sku_004 | 2500 | 2020-06-01 12:00:00 |

4 rows in set. Elapsed: 0.027 sec.

对于SummingMergeTree需要注意的几个重点：

- 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。其他的列按插入顺序保留第一行
- 不在一个分区的数据不会被聚合

- Summing并不会始终保证数据是聚合的，只有在数据合并的过程中聚合。数据合并的时机是 同一批次插入 或 分片合并 时才会进行聚合

MergeTree引擎族总结

MergeTree是clickhouse最为常用也是最为强大的表引擎族。这一族表引擎提供了大数据下的数据快速插入、管理以及检索的功能。

MergeTree一系列表引擎底层都使用类似于LSM树的方式提供数据的快速读写功能，这跟HBase其实是很像的。即新的数据(包括更新以及删除的数据)并不会影响原有的数据，而是会记录在一个新开辟的临时数据块中。查询时会通过版本号查询最新的一条结果。新开辟的数据需要等到后台进行数据合并时，才会进入主数据中。数据合并是在后台某个不确定的时间点进行的，当然也可以手动触发数据合并。

这一系列表引擎当中，以MergeTree为核心，确定了整个数据读写合并的机制。而扩展出来的一系列*MergeTree则大都是在数据合并的过程中定制数据合并的逻辑。例如ReplacingMergeTree主要关注合并时去重。SummingMergeTree主要关注合并时进行简单规则统计，生成统计报告。AggregatingMergeTree也是对合并逻辑进行定制。理解MergeTree的核心机制，这些扩展的表引擎就不难理解了。当然，具体的使用细节，还是参看官方文档：<https://clickhouse.com/docs/zh/engines/table-engines/mergetree-family/mergetree/>

另外的几种表引擎使用都比较简单，这里就不再过多分享了。有兴趣还是可以去参考一下官方文档，用到的时候查一下即可。

3.3 数据使用

clickhouse作为一个数据库，支持标准的SQL操作。这一章只分享一些clickhouse比较与众不同的用法。

3.3.1、数据导入导出

使用clickhouse首先需要有数据。我们之前也通过insert into语句造了一些测试数据，但是这种方式，在clickhouse中是非常不推荐的。一方面，insert语句插入数据，效率太低。clickhouse是面向海量数据进行查询分析，insert语句很难用来形成海量的数据。另一方面，clickhouse最常用的MergeTree表引擎，会将新插入的数据放到一个临时的分区当中，后续需要进行数据合并。频繁的insert操作会产生大量的临时分区，增加数据合并的性能消耗。所以，clickhouse中通常情况下都是通过数据文件进行大批量的导出导入操作来产生的。

最常用的数据导入导出方式是通过clickhouse-client客户端写入或读取csv文件来完成。

例如导出数据到csv文件：

```
1 clickhouse-client -h 127.0.0.1 --database="default" --query="select * from
  t_stock FORMAT CSV" > t_stock.csv
```

从csv文件导入数据：

```
1 clickhouse-client -h 127.0.0.1 --database="default" --query="insert into
  t_stock FORMAT CSV" < ./test.csv
```

另外，官方也提供了一个clickhouse-copier工具来专门对clickhouse数据进行备份与恢复。

同时，官方也提供了大量高质量的数据集可供测试。因此我们需要将这些高质量的数据集导入到clickhouse中，这样对于学习clickhouse是非常方便高效的。

官方数据集参见：<https://clickhouse.com/docs/zh/getting-started/example-datasets/>

这里面有些非常庞大的数据集，比如GitHub Events数据集，包含了31亿行数据。数据包有75G，而clickhouse保存这些数据，需要硬盘空间超过200G。所以官方这些数据集对于测试，绝对是够用的。

学习过程中，最常用的数据集还是线上测试数据库中用到的数据，也就是Yandex.Metric Data数据集。数据集包含两张表hits_v1和visits_v1。数据集可以从官方网站上下载。参见<https://clickhouse.com/docs/zh/getting-started/example-datasets/metrica/>。

而这个官方文件的导入过程相当简单粗暴，那就是直接转移数据文件。

```
1 # 导入hits_v1表
2 tar -xvf hits_v1.tar -C /var/lib/clickhouse
3 # 导入visits_v1表
4 tar -xvf visits_v1.tar -C /var/lib/clickhouse
5 # 解压出来的文件分配给clickhouse用户 -- 可选
6 chown -R clickhouse:clickhouse /var/lib/clickhouse
7 # 重启clickhouse服务
8 clickhouse restart
```

重启完成后，就可以在clickhouse中查到一个datasets数据以及hits_v1和visits_v1两张表。hits_v1表使用的是MergeTree引擎，拥有800W+的数据。visits_v1表使用的是CollapsingMergeTree引擎，拥有160W+的数据。

实际上这个导入的过程也给我们演示了clickhouse底层数据的文件结构。clickhouse的底层文件结构相比其他数据库，也是非常的简单粗暴的。

- 元数据保存在metadata目录下。
 - datasets库对应 metadata目录下的 datasets.sql文件以及datasets目录
 - 表名 对应metadata/datasets目录下的.sql文件
- 数据保存在data目录下。
 - 表数据 就对应data目录下表名对应的文件夹中。
 - 在表的数据目录 data/datasets/hits_v1/201403-10-18_2目录中，每个列对应一个.bin文件和.mrk文件。

这里需要关注下的是metadata目录下的sql语句。例如查看hits_v1表的声明文件：/var/lib/clickhouse/metadata/datasets/hits_v1.sql，看到他的内容如下：

```
1 ATTACH TABLE hits_v1
2 (
3     WatchID UInt64,
4     ... #省略字段名
5 )
6 ENGINE = MergeTree()
7 PARTITION BY toYYYYMM(EventDate)
8 ORDER BY (CounterID, EventDate, intHash32(UserID))
9 SAMPLE BY intHash32(UserID)
10 SETTINGS index_granularity = 8192
```

可以看到，就是通过一个简单的Attach语句直接将表信息加载到数据库中的。这个Attach指令并不在磁盘上实际产生数据，而是假设数据已经在正确的目录下了。这样通过执行Attach指令，clickhouse服务就将会识别对应表的存在。通过Attach语句，也可以加载默认目录以外的目录。通常是用在服务启动时加载数据信息。

而与Attach语句对应的就是DETACH语句，是将表信息进行解绑。

3.3.2、数据修改

数据修改对应update和delete操作。clickhouse也提供了这两个操作的能力，但是在clickhouse中，对数据的修改和删除是非常“重”的操作，因为对应的目标数据需要放弃原有的分区，重建新的临时分区，然后还要进行大量的合并。在语句执行过程中，只是将原有的数据打上逻辑上的删除标记，然后新增数据放入新的分区。直到触发分区合并的时候，才会删除旧的数据。频繁的update和delete操作会加大服务器的负担。

在clickhouse中，数据变更操作被称为Mutation查询，他被作为alter指令的一部分，即对表进行变更。实际上，你会发现，在官方文档上SQL部分都没有列出UPDATE和DELETE语句。通常情况下，这类Mutation操作都要交由运维人员来完成。普通用户更多的只需要关注数据的查询与分析，尽量避免不必要的数据变更操作。

```
1  -- 删除操作
2  alter table t_stock delete where sku_id='sku_001';
3  -- 修改操作
4  alter table t_stock update total_amount=toDecimal132(2000.00,2) where id=2;
```

3.3.3、数据查询

查询是clickhouse的重头戏，clickhouse除了支持标准SQL外，还提供了非常丰富的查询功能。

在标准SQL查询这一块：

- 支持子查询
- 支持各种JOIN查询。但是不建议使用。因为JOIN操作无法使用缓存。并且clickhouse执行join操作的方式是将后面的表全部加载到内存中执行，优化不是很好。表很大时性能影响非常明显。
- 支持With关键字创建临时表。例如使用下面的语句查询当前数据库中占用磁盘空间最大的10张表

```

1 WITH
2     (
3         SELECT sum(bytes)
4         FROM system.parts
5         WHERE active
6     ) AS total_disk_usage
7 SELECT
8     (sum(bytes) / total_disk_usage) * 100 AS table_disk_usage,
9     table
10 FROM system.parts
11 GROUP BY table
12 ORDER BY table_disk_usage DESC
13 LIMIT 10

```

另外，clickhouse还提供了非常丰富的特性函数

- 表函数

表函数是clickhouse非常有特色的一类函数。顾名思义，就是可以像表一样使用的函数。

例如numbers函数，可以生成一组连续的整数，在测试时非常有用。

```

1 select * from numbers(10); --产生一组0~10的整数
2 select * from numbers(10,20); --从10开始，产生20个整数
3 select toDate('2021-01-01') + number as d from numbers(365); -- 产生2021-
   01-01 ~ 2021-12-31的日期序列

```

generateRandom函数则可以产生一组随机值。例如下面的SQL语句会产生包含a,b,c三列的一个表结果。每一列都会产生一个1到10之间的随机值。

```

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

```

最后这个2是表示随机种子。一个固定的随机种子会产生稳定的随机结果。

mysql函数允许对存储在远程MySQL服务器上的数据执行SELECT和INSERT查询。

```

1 select * from mysql("hadoop01:3306", 'testdb', 'user', 'root', 'root');

```

clickhouse中还有一系列的表函数，可以直接读取远程数据库中的数据。比如file文件，hdfs文件，jdbc数据，postgresql，甚至直接访问远程服务器上的文件等。这样非常便于将远程数据与本地数据结合使用。比如在clickhouse中存储0或1这样的字典值，而将字典的字面含义直接从mysql的字典表中读取。另外，也可以很方便的将远程数据中的数据插入到clickhouse本地，直接使用。连ETL过程都省了。

- 聚合函数

也就是group by 之后进行聚合。clickhouse提供了大量的聚合函数，除了count, min, max, sum等这些常见的聚合函数外，甚至还包括了corr皮尔逊相关系数，rankCorr斯皮尔曼相关系数，simpleLinearRegression一维线性回归等机器学习中常用的聚合函数，功能相当的丰富。具体可以查看官方文档。

rankCorr计算的是斯皮尔曼相关系数。是用来计算两个数据列的相关性。这是一个-1到1之间的值。值为1，表示两个数据列完全正相关，即第一个列中较大的值对应的第二个列中的值就越大。而-1表示两个数据列负相关，即第一个列中较大的值对应第二个列中的值就越小。例如，如果一个公司在多年发展过程中，投入越多，收入也越多，也就是投入与收入的相关系数大，那就可以认为这个公司是一个非常好的公司。在统计学中，这是一个很重要的指标。

另外，在group by操作上，clickhouse还支持with rollup\with cube\with totals，来进行统计聚合。

```
1  -- 按gourp by 的顺序，从右至左逐个去掉维度进行聚合。依次按照 (id,sku_id),(id),()分
   组，对total_amount进行求和
2  select id,sku_id,sum(total_amount) from t_stock group by id,sku_id with
   rollup;
3  -- 按照goup by 的字段，互相组合进行聚合
4  select id,sku_id,sum(total_amount) from t_stock group by id,sku_id with
   cube;
5  -- 只按照group by的全字段，以及所有数据一起聚合。只按照 (id,sku_id),()两个分组进行
   求和
6  select id,sku_id,sum(total_amount) from t_stock group by id,sku_id with
   totals;
```

- 函数

clickhouse的聚合函数已经如此强大，针对普通字段的函数那就更不在话下了。常用的算数函数、比较函数、逻辑函数、类型转换函数等这些就不用多说。file函数可以直接访问操作系统的文件作为一个字段。例如

```
1 | insert into table select file('a.txt') ,file('b.txt');
```

还有用于机器学习的线性回归、逻辑回归的预测函数。甚至还有NLP自然语言处理的函数。比如自动将英文中的复数转换成为单数，将wolves转换成为wolf，这样的奇葩函数。当然，这个NLP函数目前还在实验阶段。

不过比较遗憾的是，clickhouse目前还不支持自定义函数。