

什么是ClickHouse？

ClickHouse是一个用于联机分析(OLAP)的列式数据库管理系统(DBMS)。

在传统的行式数据库系统中，数据按如下顺序存储：

行	小心点	JavaEnable	标题	GoodEvent	活动时间
#0	89354350662	1	投资者关系	1	2016-05-18 05:19:20
#1	90329509958	0	联系我们	1	2016-05-18 08:10:20
#2	89953706054	1	任务	1	2016-05-18 07:38:00
#N

处于同一行中的数据总是被物理的存储在一起。

常见的行式数据库系统有： MySQL、Postgres和MS SQL Server。

在列式数据库系统中，数据按如下的顺序存储：

行:	#0	#1	#2	#N
小心点:	89354350662	90329509958	89953706054	...
JavaEnable:	1	0	1	...
标题:	投资者关系	联系我们	任务	...
GoodEvent:	1	1	1	...
活动时间:	2016-05-18 05:19:20	2016-05-18 08:10:20	2016-05-18 07:38:00	...

该示例中只展示了数据在列式数据库中数据的排列顺序。

对于存储而言，列式数据库总是将同一列的数据存储在一起，不同列的数据也总是分开存储。

常见的列式数据库有：Vertica、Paraccel (Actian Matrix, Amazon Redshift)、Sybase IQ、Exasol、Infobright、InfiniDB、MonetDB (VectorWise, Actian Vector)、LucidDB、SAP HANA、Google Dremel、Google PowerDrill、Druid、kdb+。

不同的存储方式适合不同的场景，这里的查询场景包括：进行了哪些查询，多久查询一次以及各类查询的比例；每种查询读取多少数据———行、列和字节；读取数据和写入数据之间的关系；使用的数据集大小以及如何使用本地的数据集；是否使用事务，以及它们是如何进行隔离的；数据的复制机制与数据的完整性要求；每种类型的查询要求的延迟与吞吐量等等。

系统负载越高，根据使用场景进行定制化就越重要，并且定制将会变的越精细。没有一个系统同样适用于明显不同的场景。如果系统适用于广泛的场景，在负载高的情况下，所有的场景可以会被公平但低效处理，或者高效处理一小部分场景。

OLAP场景的关键特征

- 大多数是读请求
- 数据总是以相当大的批(> 1000 rows)进行写入
- 不修改已添加的数据
- 每次查询都从数据库中读取大量的行，但是同时又仅需要少量的列
- 宽表，即每个表包含着大量的列

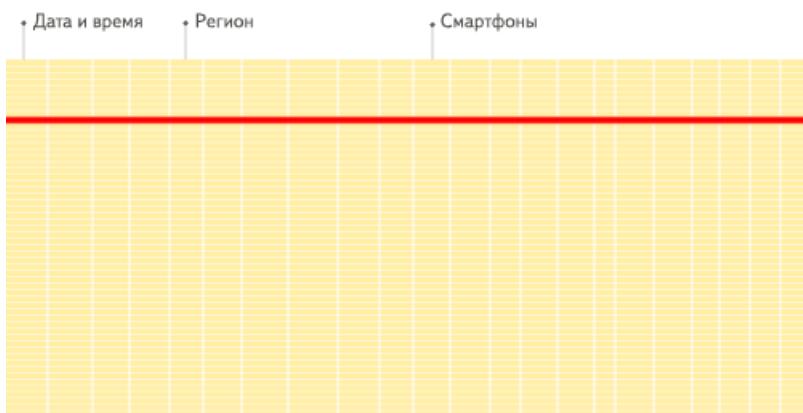
- 较少的查询(通常每台服务器每秒数百个查询或更少)
- 对于简单查询，允许延迟大约50毫秒
- 列中的数据相对较小：数字和短字符串(例如，每个URL 60个字节)
- 处理单个查询时需要高吞吐量 (每个服务器每秒高达数十亿行)
- 事务不是必须的
- 对数据一致性要求低
- 每一个查询除了一个大表外都很小
- 查询结果明显小于源数据，换句话说，数据被过滤或聚合后能够被盛放在单台服务器的内存中

很容易可以看出，OLAP场景与其他流行场景(例如,OLTP或K/V)有很大的不同，因此想要使用OLTP或Key-Value数据库去高效的处理分析查询是没有意义的，例如，使用OLAP数据库去处理分析请求通常要优于使用MongoDB或Redis去处理分析请求。

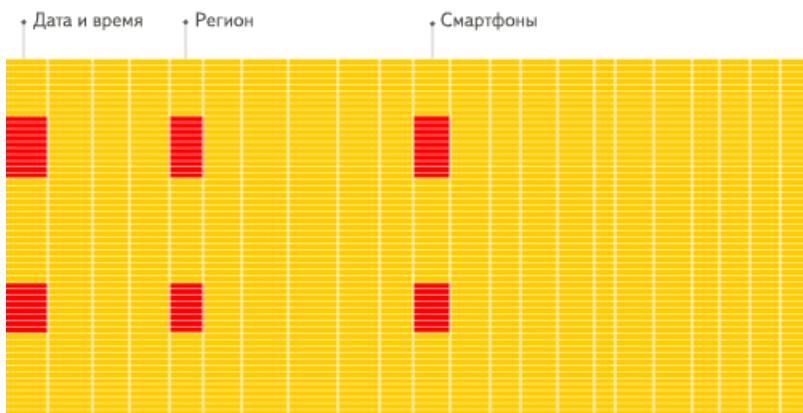
列式数据库更适合OLAP场景的原因

列式数据库更适合于OLAP场景(对于大多数查询而言，处理速度至少提高了100倍)，下面详细解释了原因(通过图片更有利子直观理解)：

行式



列式



看到差别了么？下面将详细介绍为什么会发生这种情况。

输入/输出

1. 针对分析类查询，通常只需要读取表的一小部分列。在列式数据库中你可以只读取你需要的数据。例如，如果只需要读取100列中的5列，这将帮助你最少减少20倍的I/O消耗。
2. 由于数据总是打包成批量读取的，所以压缩是非常容易的。同时数据按列分别存储这也更容易压缩。这进一步降低了I/O的体积。
3. 由于I/O的降低，这将帮助更多的数据被系统缓存。

例如，查询«统计每个广告平台的记录数量»需要读取«广告平台ID»这一列，它在未压缩的情况下需要1个字节进行存储。如果大部分流量不是来自广告平台，那么这一列至少可以以十倍的压缩率被压缩。当采用快速压缩算法，它的解压速度最少在十亿字节(未压缩数据)每秒。换句话说，这个查询可以在单个服务器上以每秒大约几十亿行的速度进行处理。这实际上是当前实

现的速度。

CPU

由于执行一个查询需要处理大量的行，因此在整个向量上执行所有操作将比在每一行上执行所有操作更加高效。同时这将有助于实现一个几乎没有调用成本的查询引擎。如果你不这样做，使用任何一个机械硬盘，查询引擎都不可避免的停止CPU进行等待。所以，在数据按列存储并且按列执行是很有意义的。

有两种方法可以做到这一点：

1. 向量引擎：所有的操作都是为向量而不是为单个值编写的。这意味着多个操作之间的不再需要频繁的调用，并且调用的成本基本可以忽略不计。操作代码包含一个优化的内部循环。
2. 代码生成：生成一段代码，包含查询中的所有操作。

这是不应该在一个通用数据库中实现的，因为这在运行简单查询时是没有意义的。但是也有例外，例如，MemSQL使用代码生成来减少处理SQL查询的延迟(只是为了比较，分析型数据库通常需要优化的是吞吐而不是延迟)。

请注意，为了提高CPU效率，查询语言必须是声明型的(SQL或MDX)，或者至少一个向量(J, K)。查询应该只包含隐式循环，允许进行优化。

ClickHouse用户

免责声明

如下使用ClickHouse的公司和他们的成功案例来源于公开资源，因此和实际情况可能有所出入。如果您分享您公司使用ClickHouse的故事，我们将不胜感激将其添加到列表，但请确保你这样做不会有任何保密协议的问题。也欢迎提供来自其他公司的出版物的更新。

公司简介	行业	用例	群集大小	(Un)压缩数据大小* (<u>of single replica</u>)	参考资料
2gis	地图	监测	—	—	俄文，2019年7月
Aloha 浏览器	移动应用程序	浏览器后端	—	—	俄文幻灯片，2019年5月
阿玛迪斯	旅行	分析	—	—	新闻稿,四月2018
Appsflyer	移动分析	主要产品	—	—	俄文，2019年7月
ArenaData	数据平台	主要产品	—	—	俄文幻灯片，十二月2019
Badoo	约会	时间序列	—	—	俄文幻灯片，十二月2019
Benocs	网络遥测和分	主要产品	—	—	英文幻灯片，2017年10月

公司简介	行业	用例	群集大小	(Un)压缩数据大小 <u>(of single replica)</u>	参考资料
彭博社	金融、媒体	监测	102个服务器	—	幻灯片，2018年5月
Bloxy	区块链	分析	—	—	俄文幻灯片，八月2018
Dataliance/UltraPower	电信	分析	—	—	中文幻灯片，2018年1月
CARTO	商业智能	地理分析	—	—	地理空间处理与ClickHouse
CERN	研究	实验	—	—	新闻稿,四月2012
思科	网络	流量分析	—	—	闪电对话，十月2019
城堡证券	金融	—	—	—	贡献，2019年3月
Citymobil	出租车	分析	—	—	俄文博客文章，三月2020
内容广场	网站分析	主要产品	—	—	法文博客文章，十一月2018
Cloudflare	CDN	流量分析	36服务器	—	博客文章,五月2017,博客文章,三月2018
Corunet	分析	主要产品	—	—	英文幻灯片，2019年4月
CreditX 氚信	金融AI	分析	—	—	英文幻灯片，2019年11月
Criteo/Storetail	零售	主要产品	—	—	英文幻灯片，十月2018
德意志银行	金融	商业智能分析	—	—	英文幻灯片，十月2019
Diva-e	数字咨询	主要产品	—	—	英文幻灯片，2019年9月
Exness	交易	指标，日志记录	—	—	俄语交谈，2019年5月
精灵	广告网络	主要产品	—	—	日文博客，2017年7月

公司简介	行业流	用例	群集大小	(Un)压缩数据大小 <u>(of single replica)</u>	参考资料片, 2018年10月
Idealista	房地产	分析	—	—	英文博客文章, 四月2019
Infovista	网络	分析	—	—	英文幻灯片, 十月2019
InnoGames	游戏	指标, 日志记录	—	—	俄文幻灯片, 2019年9月
Integros	视频服务平台	分析	—	—	俄文幻灯片, 2019年5月
科迪亚克数据	云	主要产品	—	—	虏茅驴麓卤戮碌禄路戮鲁拢
Kontur	软件开发	指标	—	—	俄语交谈, 2018年11月
LifeStreet	广告网络	主要产品	75台服务器 (3个副本)	5.27PiB	俄文博客文章, 2017年2月
Mail.ru 云解决方案	云服务	主要产品	—	—	运行ClickHouse实例, 俄语
MessageBird	电信	统计	—	—	英文幻灯片, 2018年11月
MGID	广告网络	网络分析	—	—	我们在实施分析DBMS ClickHouse的经验, 俄文
OneAPM	监测和数据分析	主要产品	—	—	中文幻灯片, 2018年10月
Pragma Innovation	遥测和大数据分析	主要产品	—	—	英文幻灯片, 十月2018
青云	云服务	主要产品	—	—	中文幻灯片, 2018年10月
Qrator	DDoS保护	主要产品	—	—	博客文章, 三月2019
百分点	分析	主要产品	—	—	中文幻灯片, 2019年6月
...

漫步者 公司简介	互联网 行业	分析 用例	—群集大小	— (Un)压缩数据大 小 (of single replica)	俄语讲座，2018年4 月 参考资料
腾讯	通讯软 件	日志 记录	—	— replica)	中文讲座，2019年 11月
流量之星	广告网 络	—	—	—	俄文幻灯片，2018年 5月
S7航空公司	航空公 司	指 标， 日志 记录	—	—	俄文，2019年3月
SEMrush	营销	主要 产品	—	—	俄文幻灯片，八月 2018
scireum GmbH	电子商 务	主要 产品	—	—	德语讲座，2020年2 月
Sentry	软件开 发	产品 后端	—	—	英文博客文章,五月 2019
SGK	政府社 会保障	分析	—	—	英文幻灯片，2019年 11月
seo.do	分析	主要 产品	—	—	英文幻灯片，2019年 11月
新浪	新闻	—	—	—	中文幻灯片，2018年 10月
SMI2	新闻	分析	—	—	俄文博客文章，2017 年11月
Splunk	业务分 析	主要 产品	—	—	英文幻灯片，2018年 1月
Spotify	音乐	实验	—	—	幻灯片，七月2018
腾讯	大数据	数据 处理	—	—	中文幻灯片，2018年 10月
优步	出租车	日志 记录	—	—	幻灯片，二月2020
VKontakte	社交网 络	统 计， 日志 记录	—	—	俄文幻灯片，八月 2018
Wisebits	IT解决 方案	分析	—	—	俄文幻灯片，2019年 5月

公司简介	行业	共同用例目的	群集大小	(Un)压缩数据大小 <u>(of single replica)</u>	英文幻灯片，2019年11月 参考资料
喜马拉雅	音频共享	OLAP	—	—	英文幻灯片，2019年11月
Yandex云	公有云	主要产品	—	—	俄文，2019年12月
Yandex DataLens	商业智能	主要产品	—	—	俄文幻灯片，十二月2019
Yandex市场	电子商务	指标，日志记录	—	—	俄文，2019年1月
Yandex Metrica	网站分析	主要产品	一个集群中的360台服务器，一个部门中的1862台服务器	66.41PiB/5.68PiB	幻灯片，二月2020
ЦВТ	软件开发	指标，日志记录	—	—	博客文章，三月2019，俄文
МКБ	银行	网络系统监控	—	—	俄文幻灯片，2019年9月
金数据	商业智能分析	主要产品	—	—	中文幻灯片，2019年10月
Instana	APM平台	主要产品	—	—	推特消息
Wargaming	游戏		—	—	采访
CrazyPanda	游戏		—	—	ClickHouse 社区会议
FunCorp	游戏		—	—	文章

ClickHouse历史

ClickHouse最初是为 YandexMetrica 世界第二大Web分析平台 而开发的。多年来一直作为该系统的核心组件被该系统持续使用着。目前为止，该系统在ClickHouse中有超过13万亿条记录，并且每天超过200多亿个事件被处理。它允许直接从原始数据中动态查询并生成报告。本文简要介绍了ClickHouse在其早期发展阶段的目标。

Yandex.Metrica基于用户定义的字段，对实时访问、连接会话，生成实时的统计报表。这种需求往往需要复杂聚合方式，比如对访问用户进行去重。构建报表的数据，是实时接收存储的新数据。

截至2014年4月，Yandex.Metrica每天跟踪大约120亿个事件（用户的点击和浏览）。为了可以创建自定义的报表，我们必须存储全部这些事件。同时，这些查询可能需要在几百毫秒内扫描数百万行的数据，或在几秒内扫描数亿行的数据。

Yandex.Metrica以及其他Yandex服务的使用案例

在Yandex.Metrica中，ClickHouse被用于多个场景中。

它的主要任务是使用原始数据在线的提供各种数据报告。它使用374台服务器的集群，存储了20.3万亿行的数据。在去除重复与副本数据的情况下，压缩后的数据达到了2PB。未压缩前（TSV格式）它大概有17PB。

ClickHouse还被使用在：

- 存储来自Yandex.Metrica回话重放数据。
- 处理中间数据
- 与Analytics一起构建全球报表。
- 为调试Yandex.Metrica引擎运行查询
- 分析来自API和用户界面的日志数据

ClickHouse在其他Yandex服务中至少有12个安装：search verticals, Market, Direct, business analytics, mobile development, AdFox, personal services等。

聚合与非聚合数据

有一种流行的观点认为，想要有效的计算统计数据，必须要聚合数据，因为聚合将降低数据量。

但是数据聚合是一个有诸多限制的解决方案，例如：

- 你必须提前知道用户定义的报表的字段列表
- 用户无法自定义报表
- 当聚合条件过多时，可能不会减少数据，聚合是无用的。
- 存在大量报表时，有太多的聚合变化（组合爆炸）
- 当聚合条件有非常大的基数时（如：url），数据量没有太大减少（少于两倍）
- 聚合的数据量可能会增长而不是收缩
- 用户不会查看我们为他生成的所有报告，大部分计算将是无用的
- 各种聚合可能违背了数据的逻辑完整性

如果我们直接使用非聚合数据而不进行任何聚合时，我们的计算量可能是减少的。

然而，相对于聚合中很大一部分工作被离线完成，在线计算需要尽快的完成计算，因为用户在等待结果。

Yandex.Metrica有一个专门用于聚合数据的系统，称为Metrage，它可以用作大部分报表。

从2009年开始，Yandex.Metrica还为非聚合数据使用专门的OLAP数据库，称为OLAPServer，它以前用于报表构建系统。OLAPServer可以很好的工作在非聚合数据上，但是它有诸多限制，导致无法根据需要将其用于所有报表中。如，缺少对数据类型的支持（只支持数据），无法实时增量的更新数据（只能通过每天重写数据完成）。OLAPServer不是一个数据库管理系统，它只是一个数据库。

为了消除OLAPServer的这些局限性，解决所有报表使用非聚合数据的问题，我们开发了ClickHouse数据库管理系统。

ClickHouse的特性

真正的列式数据库管理系统

在一个真正的列式数据库管理系统中，除了数据本身外不应该存在其他额外的数据。这意味着为了避免在值旁边存储它们的长度«number»，你必须支持固定长度数值类型。例如，10亿个UInt8类型的数据在未压缩的情况下大约消耗1GB左右的空间，如果不是这样的话，这将对CPU的使用产生强烈影响。即使是在未压缩的情况下，紧凑的存储数据也是非常重要的，因为解压缩的速度主要取决于未压缩数据的大小。

这是非常值得注意的，因为在一些其他系统中也可以将不同的列分别进行存储，但由于对其他场景进行的优化，使其无法有效的处理分析查询。例如：HBase，BigTable，Cassandra，HyperTable。在这些系统中，你可以得到每秒数十万的吞吐能力，但是无法得到每秒几亿行的吞吐能力。

需要说明的是，ClickHouse不单单是一个数据库，它是一个数据库管理系统。因为它允许在运行时创建表和数据库、加载数据和运行查询，而无需重新配置或重启服务。

数据压缩

在一些列式数据库管理系统中(例如：`InfiniDB CE` 和 `MonetDB`) 并没有使用数据压缩。但是, 若想达到比较优异的性能, 数据压缩确实起到了至关重要的作用。

数据的磁盘存储

许多的列式数据库(如 `SAP HANA`, `Google PowerDrill`)只能在内存中工作, 这种方式会造成比实际更多的设备预算。`ClickHouse`被设计用于工作在传统磁盘上的系统, 它提供每GB更低的存储成本, 但如果有可能使用SSD和内存, 它也会合理的利用这些资源。

多核心并行处理

`ClickHouse`会使用服务器上一切可用的资源, 从而以最自然的方式并行处理大型查询。

多服务器分布式处理

上面提到的列式数据库管理系统中, 几乎没有一个支持分布式的查询处理。

在`ClickHouse`中, 数据可以保存在不同的shard上, 每一个shard都由一组用于容错的replica组成, 查询可以并行地在所有shard上进行处理。这些对用户来说是透明的

支持SQL

`ClickHouse`支持基于SQL的声明式查询语言, 该语言大部分情况下是与SQL标准兼容的。

支持的查询包括 `GROUP BY`, `ORDER BY`, `IN`, `JOIN`以及非相关子查询。

不支持窗口函数和相关子查询。

向量引擎

为了高效的使用CPU, 数据不仅仅按列存储, 同时还按向量(列的一部分)进行处理, 这样可以更加高效地使用CPU。

实时的数据更新

`ClickHouse`支持在表中定义主键。为了使查询能够快速在主键中进行范围查找, 数据总是以增量的方式有序的存储在MergeTree中。因此, 数据可以持续不断地高效的写入到表中, 并且写入的过程中不会存在任何加锁的行为。

索引

按照主键对数据进行排序, 这将帮助`ClickHouse`在几十毫秒以内完成对数据特定值或范围的查找。

适合在线查询

在线查询意味着在没有对数据做任何预处理的情况下以极低的延迟处理查询并将结果加载到用户的页面中。

支持近似计算

`ClickHouse`提供各种各样在允许牺牲数据精度的情况下对查询进行加速的方法:

1. 用于近似计算的各类聚合函数, 如:`distinct values`, `medians`, `quantiles`
2. 基于数据的部分样本进行近似查询。这时, 仅会从磁盘检索少部分比例的数据。
3. 不使用全部的聚合条件, 通过随机选择有限个数据聚合条件进行聚合。这在数据聚合条件满足某些分布条件下, 在提供相当准确的聚合结果的同时降低了计算资源的使用。

支持数据复制和数据完整性

`ClickHouse`使用异步的多主复制技术。当数据被写入任何一个可用副本后, 系统会在后台将数据分发给其他副本, 以保证系统在不同副本上保持相同的数据。在大多数情况下`ClickHouse`能在故障后自动恢复, 在一些少数的复杂情况下需要手动恢复。

更多信息, 参见 [数据复制](#)。

限制

1. 没有完整的事务支持。

2. 缺少高频率，低延迟的修改或删除已存在数据的能力。仅能用于批量删除或修改数据，但这符合 **GDPR**。

3. 稀疏索引使得ClickHouse不适合通过其键检索单行的点查询。

性能

根据Yandex的内部测试结果，ClickHouse表现出了比同类可比较产品更优的性能。你可以在 [这里](#) 查看具体的测试结果。

许多其他的测试也证实这一点。你可以使用互联网搜索到它们，或者你也可以从 [我们收集的部分相关连接](#) 中查看。

单个大查询的吞吐量

吞吐量可以使用每秒处理的行数或每秒处理的字节数来衡量。如果数据被放置在page cache中，则一个不太复杂的查询在单个服务器上大约能够以 $2\text{-}10\text{GB}/\text{s}$ （未压缩）的速度进行处理（对于简单的查询，速度可以达到 $30\text{GB}/\text{s}$ ）。如果数据没有在page cache中的话，那么速度将取决于你的磁盘系统和数据的压缩率。例如，如果一个磁盘允许以 $400\text{MB}/\text{s}$ 的速度读取数据，并且数据压缩率是3，则数据的处理速度为 $1.2\text{GB}/\text{s}$ 。这意味着，如果你是在提取一个10字节的列，那么它的处理速度大约是1-2亿行每秒。

对于分布式处理，处理速度几乎是线性扩展的，但这受限于聚合或排序的结果不是那么大的情况下。

处理短查询的延迟时间

如果一个查询使用主键并且没有太多行(几十万)进行处理，并且没有查询太多的列，那么在数据被page cache缓存的情况下，它的延迟应该小于50毫秒(在最佳的情况下应该小于10毫秒)。否则，延迟取决于数据的查找次数。如果你当前使用的是HDD，在数据没有加载的情况下，查询所需要的延迟可以通过以下公式计算得知： 查找时间 (10 ms) * 查询的列的数量 * 查询的数据块的数量。

处理大量短查询的吞吐量

在相同的情况下，ClickHouse可以在单个服务器上每秒处理数百个查询（在最佳的情况下最多可以处理数千个）。但是由于这不适用于分析型场景。因此我们建议每秒最多查询100次。

数据的写入性能

我们建议每次写入不少于1000行的批量写入，或每秒不超过一个写入请求。当使用tab-separated格式将一份数据写入到MergeTree表中时，写入速度大约为50到200MB/s。如果您写入的数据每行为1Kb，那么写入的速度为50,000到200,000行每秒。如果您的行更小，那么写入速度将更高。为了提高写入性能，您可以使用多个INSERT进行并行写入，这将带来线性的性能提升。

懒惰

仅将表保留在RAM中 `expiration_time_in_seconds` 上次访问后几秒钟。只能与*日志表一起使用。

它针对存储许多小*日志表进行了优化，访问之间存在较长的时间间隔。

创建数据库

```
CREATE DATABASE testlazy ENGINE = Lazy(expiration_time_in_seconds);
```

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数据库引擎会将对其的查询转换为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。

但您无法对其执行以下操作：

- RENAME
- CREATE TABLE
- ALTER

CREATE DATABASE

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

MySQL数据库引擎参数

- 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。

支持的类型对应

MySQL	ClickHouse
UNSIGNED TINYINT	UInt8
TINYINT	Int8
UNSIGNED SMALLINT	UInt16
SMALLINT	Int16
UNSIGNED INT, UNSIGNED MEDIUMINT	UInt32
INT, MEDIUMINT	Int32
UNSIGNED BIGINT	UInt64
BIGINT	Int64
FLOAT	Float32
DOUBLE	Float64
DATE	日期
DATETIME, TIMESTAMP	日期时间
BINARY	固定字符串

其他的MySQL数据类型将全部都转换为字符串。

同时以上的所有类型都支持可为空。

使用示例

在MySQL中创建表：

```
mysql> USE test;
Database changed

mysql> CREATE TABLE `mysql_table` (
    -> `int_id` INT NOT NULL AUTO_INCREMENT,
    -> `float` FLOAT NOT NULL,
    -> PRIMARY KEY (`int_id`));
Query OK, 0 rows affected (0,09 sec)

mysql> insert into mysql_table(`int_id`, `float`) VALUES (1,2);
Query OK, 1 row affected (0,00 sec)

mysql> select * from mysql_table;
+-----+-----+
| int_id | value |
+-----+-----+
|     1 |     2 |
+-----+-----+
1 row in set (0,00 sec)
```

在ClickHouse中创建MySQL类型的数据表，同时与MySQL服务器交换数据：

```
CREATE DATABASE mysql_db ENGINE = MySQL('localhost:3306', 'test', 'my_user', 'user_password')
```

```
SHOW DATABASES
```

```
name
+-----+
| default |
| mysql_db |
| system   |
+-----+
```

```
SHOW TABLES FROM mysql_db
```

```
name
+-----+
| mysql_table |
+-----+
```

```
SELECT * FROM mysql_db.mysql_table
```

```
+-----+-----+
| int_id | value |
+-----+-----+
|     1 |     2 |
+-----+-----+
```

```
INSERT INTO mysql_db.mysql_table VALUES (3,4)
```

```
SELECT * FROM mysql_db.mysql_table
```

int_id	value
1	2
3	4

数据库引擎

您使用的所有表都是由数据库引擎所提供的

默认情况下，ClickHouse使用自己的数据库引擎，该引擎提供可配置的表引擎和所有支持的SQL语法。

除此之外，您还可以选择使用以下的数据引擎：

- MySQL

版本集合在新树

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。

请参阅部分 [崩溃](#) 有关详细信息。

引擎继承自 [MergeTree](#) 并将折叠行的逻辑添加到合并数据部分的算法中。[VersionedCollapsingMergeTree](#) 用于相同的目的 [折叠树](#) 但使用不同的折叠算法，允许以多个线程的任何顺序插入数据。特别是，[Version](#) 列有助于正确折叠行，即使它们以错误的顺序插入。相比之下，[CollapsingMergeTree](#) 只允许严格连续插入。

创建表

```
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, ...]
```

有关查询参数的说明，请参阅 [查询说明](#)。

发动机参数

[VersionedCollapsingMergeTree\(sign, version\)](#)

- sign — Name of the column with the type of row: 1 是一个“state”行，-1 是一个“cancel”行

列数据类型应为 [Int8](#)。

- `version` — Name of the column with the version of the object state.

列数据类型应为 `UInt*`.

查询子句

当创建一个 `VersionedCollapsingMergeTree` 表，相同 [条款](#) 需要创建一个时 `MergeTree` 桌子

- 不推荐使用的创建表的方法

崩溃

数据

考虑一种情况，您需要为某个对象保存不断变化的数据。对于一个对象有一行，并在发生更改时更新该行是合理的。但是，对于数据库管理系统来说，更新操作非常昂贵且速度很慢，因为它需要重写存储中的数据。如果需要快速写入数据，则不能接受更新，但可以按如下顺序将更改写入对象。

使用 `Sign` 列写入行时。如果 `Sign = 1` 这意味着该行是一个对象的状态（让我们把它称为“state”行）。如果 `Sign = -1` 它指示具有相同属性的对象的状态的取消（让我们称之为“cancel”行）。还可以使用 `Version` 列，它应该用单独的数字标识对象的每个状态。

例如，我们要计算用户在某个网站上访问了多少页面以及他们在那里的时间。在某个时间点，我们用用户活动的状态写下面的行：

<code>UserID</code>	<code>PageViews</code>	<code>Duration</code>	<code>Sign</code>	<code>Version</code>
4324182021466249494	5	146	1	1

在稍后的某个时候，我们注册用户活动的变化，并用以下两行写入它。

<code>UserID</code>	<code>PageViews</code>	<code>Duration</code>	<code>Sign</code>	<code>Version</code>
4324182021466249494	5	146	-1	1
4324182021466249494	6	185	1	2

第一行取消对象（用户）的先前状态。它应该复制已取消状态的所有字段，除了 `Sign`.

第二行包含当前状态。

因为我们只需要用户活动的最后一个状态，行

<code>UserID</code>	<code>PageViews</code>	<code>Duration</code>	<code>Sign</code>	<code>Version</code>
4324182021466249494	5	146	1	1
4324182021466249494	5	146	-1	1

可以删除，折叠对象的无效（旧）状态。`VersionedCollapsingMergeTree` 在合并数据部分时执行此操作。

要了解为什么每次更改都需要两行，请参阅 [算法](#).

使用注意事项

1. 写入数据的程序应该记住对象的状态以取消它。该“cancel”字符串应该是“state”与相反的字符串 `Sign`. 这增加了存储的初始大小，但允许快速写入数据。
2. 列中长时间增长的数组由于写入负载而降低了引擎的效率。数据越简单，效率就越高。
3. `SELECT` 结果很大程度上取决于对象变化历史的一致性。准备插入数据时要准确。您可以通过不一致的数据获得不可预测的结果。例如从坏源读取非本地化的名称。

例如的插入，例如会插入反序升序但你的价值。

算法

当 ClickHouse 合并数据部分时，它会删除具有相同主键和版本且不同主键和版本的每对行。行的顺序并不重要。

当 ClickHouse 插入数据时，它会按主键对行进行排序。如果 `Version` 列不在主键中，ClickHouse 将其隐式添加到主键作为最后一个字段并使用它进行排序。

选择数据

ClickHouse 不保证具有相同主键的所有行都将位于相同的结果数据部分中，甚至位于相同的物理服务器上。对于写入数据和随后合并数据部分都是如此。此外，ClickHouse 流程 `SELECT` 具有多个线程的查询，并且无法预测结果中的行顺序。这意味着聚合是必需的，如果有必要得到完全“collapsed”从数据 `VersionedCollapsingMergeTree` 桌子

要完成折叠，请使用 `GROUP BY` 考虑符号的子句和聚合函数。例如，要计算数量，请使用 `sum(Sign)` 而不是 `count()`。要计算的东西的总和，使用 `sum(Sign * x)` 而不是 `sum(x)`，并添加 `HAVING sum(Sign) > 0`。

聚合 `count`, `sum` 和 `avg` 可以这样计算。聚合 `uniq` 如果对象至少具有一个非折叠状态，则可以计算。聚合 `min` 和 `max` 无法计算是因为 `VersionedCollapsingMergeTree` 不保存折叠状态值的历史记录。

如果您需要提取数据“collapsing”但是，如果没有聚合（例如，要检查是否存在其最新值与某些条件匹配的行），则可以使用 `FINAL` 修饰符 `FROM` 条款这种方法效率低下，不应与大型表一起使用。

使用示例

示例数据：

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	1	1
4324182021466249494	5	146	-1	1
4324182021466249494	6	185	1	2

创建表：

```
CREATE TABLE UAct
(
    UserID UInt64,
    PageViews UInt8,
    Duration UInt8,
    Sign Int8,
    Version UInt8
)
ENGINE = VersionedCollapsingMergeTree(Sign, Version)
ORDER BY UserID
```

插入数据：

```
INSERT INTO UAct VALUES (4324182021466249494, 5, 146, 1, 1)
```

```
INSERT INTO UAct VALUES (4324182021466249494, 5, 146, -1, 1), (4324182021466249494, 6, 185, 1, 2)
```

我们用两个 `INSERT` 查询以创建两个不同的数据部分。如果我们使用单个查询插入数据，ClickHouse 将创建一个数据部分，并且永远不会执行任何合并。

获取数据：

```
SELECT * FROM UAct
```

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	1	1

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	-1	1
4324182021466249494	6	185	1	2

我们在这里看到了什么，折叠的部分在哪里？

我们使用两个创建了两个数据部分 `INSERT` 查询。该 `SELECT` 查询是在两个线程中执行的，结果是行的随机顺序。由于数据部分尚未合并，因此未发生折叠。ClickHouse在我们无法预测的未知时间点合并数据部分。

这就是为什么我们需要聚合：

```
SELECT
    UserID,
    sum(PageViews * Sign) AS PageViews,
    sum(Duration * Sign) AS Duration,
    Version
FROM UAct
GROUP BY UserID, Version
HAVING sum(Sign) > 0
```

UserID	PageViews	Duration	Version
4324182021466249494	6	185	2

如果我们不需要聚合，并希望强制折叠，我们可以使用 `FINAL` 修饰符 `FROM` 条款

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign	Version
4324182021466249494	6	185	1	2

这是一个非常低效的方式来选择数据。不要把它用于大桌子。

GraphiteMergeTree

此引擎专为细化和聚合/平均 (rollup) 石墨 戴达 对于想要使用ClickHouse作为Graphite的数据存储的开发人员来说，这可能会有所帮助。

您可以使用任何ClickHouse表引擎来存储石墨数据，如果你不需要汇总，但如果你需要一个汇总使用 `GraphiteMergeTree`。该引擎减少了存储量，并提高了Graphite查询的效率。

引擎继承从属性 `MergeTree`.

创建表

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    Path String,
    Time DateTime,
    Value <Numeric_type>,
    Version <Numeric_type>
    ...
) ENGINE = GraphiteMergeTree(config_section)
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

请参阅的详细说明 [CREATE TABLE](#) 查询。

Graphite数据的表应具有以下数据的列:

- 公制名称（石墨传感器）。 数据类型: `String`.
- 测量度量的时间。 数据类型: `DateTime`.
- 度量值。 数据类型：任何数字。
- 指标的版本。 数据类型：任何数字。

如果版本相同，ClickHouse会保存版本最高或最后写入的行。其他行在数据部分合并期间被删除。

应在汇总配置中设置这些列的名称。

GraphiteMergeTree参数

- `config_section` — Name of the section in the configuration file, where are the rules of rollup set.

查询子句

当创建一个 `GraphiteMergeTree` 表，相同 [条款](#) 是必需的，因为当创建 `MergeTree` 桌子

► 不推荐使用的创建表的方法

汇总配置

汇总的设置由 [graphite_rollup](#) 服务器配置中的参数。参数的名称可以是any。您可以创建多个配置并将它们用于不同的表。

汇总配置结构:

```
required-columns
patterns
```

必填列

- `path_column_name` — The name of the column storing the metric name (Graphite sensor). Default value: `Path`.
- `time_column_name` — The name of the column storing the time of measuring the metric. Default value: `Time`.
- `value_column_name` — The name of the column storing the value of the metric at the time set in `time_column_name`. 默认值: `Value`.
- `version_column_name` — The name of the column storing the version of the metric. Default value: `Timestamp`.

模式

的结构 `patterns` 科:

```
pattern
  regexp
  function
pattern
  regexp
  age + precision
...
pattern
  regexp
  function
  age + precision
...
pattern
...
default
  function
  age + precision
...
```

注意

模式必须严格排序:

1. Patterns without `function` or `retention`.
2. Patterns with both `function` and `retention`.
3. Pattern `default`.

在处理行时，ClickHouse会检查以下内容中的规则 `pattern` 部分。每个 `pattern`（包括 `default`）部分可以包含 `function` 聚合参数, `retention` 参数或两者兼而有之。如果指标名称匹配 `regexp`，从规则 `pattern` 部分（sections节）的应用;否则，从规则 `default` 部分被使用。

字段为 `pattern` 和 `default` 科:

- `regexp` - A pattern for the metric name.
- `age` - The minimum age of the data in seconds.
- `precision` - How precisely to define the age of the data in seconds. Should be a divisor for 86400 (seconds in a day).
- `function` - The name of the aggregating function to apply to data whose age falls within the range [`age`, `age + precision`].

配置示例

```
<graphite_rollup>
<version_column_name>Version</version_column_name>
<pattern>
  <regexp>click_cost</regexp>
  <function>any</function>
  <retention>
    <age>0</age>
    <precision>5</precision>
  </retention>
  <retention>
    <age>86400</age>
    <precision>60</precision>
```

```
<precision>60</precision>
</retention>
</pattern>
<default>
    <function>max</function>
    <retention>
        <age>0</age>
        <precision>60</precision>
    </retention>
    <retention>
        <age>3600</age>
        <precision>300</precision>
    </retention>
    <retention>
        <age>86400</age>
        <precision>3600</precision>
    </retention>
</default>
</graphite_rollup>
```

AggregatingMergeTree

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。

建表

```
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]
[SETTINGS name=value, ...]
```

语句参数的说明，请参阅 [语句描述](#)。

子句

创建 `AggregatingMergeTree` 表时，需用跟创建 `MergeTree` 表一样的 [子句](#)。

► 已弃用的建表方法

SELECT 和 INSERT

插入数据，需使用带有聚合 -State- 函数的 [INSERT SELECT](#) 语句。

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`。

在 `SELECT` 查询的结果中，对于 ClickHouse 的所有输出格式 `AggregateFunction` 类型的值都实现了特定的二进制表示法。

如果直接用 `SELECT` 取出这些数据，然后使用 `TOJSON` 函数，那么这些值会根据此直接用 `TOJSON` 函数上转换。

如果直接用 `SELECT` 子出这些数据，例如如用 `TabSeparated` 格式，那么这些子出数据也能直接用 `INSERT` 命令加载子入。

聚合物化视图的示例

创建一个跟踪 `test.visits` 表的 `AggregatingMergeTree` 物化视图：

```
CREATE MATERIALIZED VIEW test.basic
ENGINE = AggregatingMergeTree() PARTITION BY toYYYYMM(StartDate) ORDER BY (CounterID, StartDate)
AS SELECT
    CounterID,
    StartDate,
    sumState(Sign) AS Visits,
    uniqState(UserID) AS Users
FROM test.visits
GROUP BY CounterID, StartDate;
```

向 `test.visits` 表中插入数据。

```
INSERT INTO test.visits ...
```

数据会同时插入到表和视图中，并且视图 `test.basic` 会将里面的数据聚合。

要获取聚合数据，我们需要在 `test.basic` 视图上执行类似 `SELECT ... GROUP BY ...` 这样的查询：

```
SELECT
    StartDate,
    sumMerge(Visits) AS Visits,
    uniqMerge(Users) AS Users
FROM test.basic
GROUP BY StartDate
ORDER BY StartDate;
```

MergeTree

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

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

主要特点：

- 存储的数据按主键排序。

这让你可以创建一个用于快速检索数据的小稀疏索引。

- 允许使用分区，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。

- 支持数据副本。

`ReplicatedMergeTree` 系列的表便是用于此。更多信息，请参阅 [数据副本](replication.md) 一节。

- 支持数据采样。

需要的话，你可以给表设置一个采样方法。

注意

合并 引擎并不属于 *MergeTree 系列。

建表

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
) ENGINE = MergeTree()
[PARTITION BY expr]
[ORDER BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

请求参数的描述，参考 [请求描述](#)。

子句

- **ENGINE** — 引擎名和参数。 `ENGINE = MergeTree()`. MergeTree 引擎没有参数。
- **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] ([../../engines/table_engines/mergetree_family/mergetree.md](#)) 类型的列。这里该分区名格式会是 `"YYYYMM"` 这样。

- **ORDER BY** — 表的排序键。

可以是一组列的元组或任意的表达式。例如: `ORDER BY (CounterID, EventDate)`。

- **PRIMARY KEY** — 主键，如果要设成 [跟排序键不相同](#)。

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。

- **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))'`。

- **SETTINGS** — 影响 MergeTree 性能的额外参数：

- `index_granularity` — 索引粒度。即索引中相邻『标记』间的数据行数。默认值，8192。该列表中所有可用的参数可以从[这里查看 MergeTreeSettings.h](#)。
- `index_granularity_bytes` — 索引粒度，以字节为单位，默认值: 10Mb。如果仅按数据行数限制索引粒度，请设置

为0(不建议)。

- `enable_mixed_granularity_parts` — 启用或禁用通过 `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。当从大表(数十或数百兆)中查询数据时候，`index_granularity_bytes` 配置能够提升ClickHouse的性能。如果你的表内数据量很大，可以开启这项配置用以提升 `SELECT` 查询的性能。
- `use_minimalistic_part_header_in_zookeeper` — 数据片段头在 ZooKeeper 中的存储方式。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。默认值：`86400` (1 天)。
- `write_final_mark` — 启用或禁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)。
- `storage_policy` — 存储策略。参见 [使用多个区块装置进行数据存储](#).

示例配置

```
ENGINE MergeTree() PARTITION BY toYYYYMM(EventDate) ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID) SETTINGS index_granularity=8192
```

示例中，我们设为按月分区。

同时我们设置了一个按用户 ID 哈希的抽样表达式。这让你可以有该表中每个 `CounterID` 和 `EventDate` 下面的数据的伪随机分布。如果你在查询时指定了 `SAMPLE` 子句。ClickHouse 会返回对于用户子集的一个均匀的伪随机数据采样。

`index_granularity` 可省略，默认值为 `8192`。

▶ 已弃用的建表方法

数据存储

表由按主键排序的数据 片段 组成。

当数据被插入到表中时，会分成数据片段并按主键的字典序排序。例如，主键是 `(CounterID, Date)` 时，片段中数据按 `CounterID` 排序，具有相同 `CounterID` 的部分按 `Date` 排序。

不同分区的数据会被分成不同的片段，ClickHouse 在后台合并数据片段以便更高效存储。不会合并来自不同分区的数据片段。这个合并机制并不保证相同主键的所有行都会合并到同一个数据片段中。

ClickHouse 会为每个数据片段创建一个索引文件，索引文件包含每个索引行（『标记』）的主键值。索引行号定义为 `n * index_granularity`。最大的 `n` 等于总行数除以 `index_granularity` 的值的整数部分。对于每列，跟主键相同的索引行处也会写入『标记』。这些『标记』让你可以直接找到数据所在的列。

你可以只用一单一大表并不断地一块块往里面加入数据 - `MergeTree` 引擎的就是为了这样的场景。

主键和索引在查询中的表现

我们以 `(CounterID, Date)` 以主键。排序好的索引的图示会是下面这样：

```
全部数据 : [-----]
CounterID: [aaaaaaaaaaaaaaaaaaaaabbbbcdeeeeeeeeefggggggghhhhhhhiiiiiiik|||||||]
Date:      [11111112222223331233211112222233211111121222222311112223311122333]
标记:      | | | | | | | | | |
a,1  a,2  a,3  b,3  e,2  e,3  g,1  h,2  i,1  i,3  l,3
标记号:    0   1   2   3   4   5   6   7   8   9   10
```

如果指定查询如下：

- CounterID in ('a', 'h')，服务器会读取标记号在 [0, 3] 和 [6, 8] 区间中的数据。
- CounterID IN ('a', 'h') AND Date = 3，服务器会读取标记号在 [1, 3] 和 [7, 8] 区间中的数据。
- Date = 3，服务器会读取标记号在 [1, 10] 区间中的数据。

上面例子可以看出使用索引通常会比全表描述要高效。

稀疏索引会引起额外的数据读取。当读取主键单个区间范围的数据时，每个数据块中最多会多读 `index_granularity * 2` 行额外的数据。大部分情况下，当 `index_granularity = 8192` 时，ClickHouse的性能并不会降级。

稀疏索引让你能操作有巨量行的表。因为这些索引是常驻内存（RAM）的。

ClickHouse 不要求主键惟一。所以，你可以插入多条具有相同主键的行。

主键的选择

主键中列的数量并没有明确的限制。依据数据结构，你应该让主键包含多些或少些列。这样可以：

- 改善索引的性能。

如果当前主键是 `(a, b)`，然后加入另一个 `c` 列，满足下面条件时，则可以改善性能：

- 有带有 `c` 列条件的查询。
- 很长的数据范围（`index_granularity` 的数倍）里 `(a, b)` 都是相同的值，并且这种情况很普遍。换言之，就是加入另一列后，可以让你的查询略过很长的数据范围。

- 改善数据压缩。

ClickHouse 以主键排序片段数据，所以，数据的一致性越高，压缩越好。

- 折叠树 和 SummingMergeTree 引擎里，数据合并时，会有额外的处理逻辑。

在这种情况下，指定一个跟主键不同的 *排序键* 也是有意义的。

长的主键会对插入性能和内存消耗有负面影响，但主键中额外的列并不影响 `SELECT` 查询的性能。

选择跟排序键不一样主键

指定一个跟排序键（用于排序数据片段中行的表达式）

不一样的主键（用于计算写到索引文件的每个标记值的表达式）是可以的。

这种情况下，主键表达式元组必须是排序键表达式元组的一个前缀。

当使用 SummingMergeTree 和

AggregatingMergeTree 引擎时，这个特性非常有用。

通常，使用这类引擎时，表里列分两种：维度和度量。

典型的查询是在 `GROUP BY` 并过滤维度的情况下统计度量列的值。

像 SummingMergeTree 和 AggregatingMergeTree，用相同的排序键值统计行时，通常会加上所有的维度。结果就是，这键的表达式会是一长串的列组成，

并且这组列还会因为新加维度必须频繁更新。

这种情况下，主键中仅预留少量列保证高效范围扫描，

剩下的维度列放到排序键元组里。这样是合理的。

排序键的修改 是轻量级的操作，因为一个新列同时被加入到表里和排序键后时，已存在的数据片段并不需要修改。由于旧的排序键是新排序键的前缀，并且刚刚添加的列中没有数据，因此在表修改时的数据对于新旧的排序键来说都是有序的。

索引和分区在查询中的应用

对于 `SELECT` 查询，ClickHouse 分析是否可以使用索引。如果 `WHERE/PREWHERE` 子句具有下面这些表达式（作为谓词链接一子项或整个）则可以使用索引：基于主键或分区键的列或表达式的部分的等式或比较运算表达式；基于主键或分区键的列或表达式的固定前缀的 `IN` 或 `LIKE` 表达式；基于主键或分区键的列的某些函数；基于主键或分区键的表达式的逻辑表达式。

因此，在索引键的一个或多个区间上快速地跑查询都是可能的。下面例子中，指定标签；指定标签和日期范围；指定标签和日期；指定多个标签和日期范围等运行查询，都会非常快。

当引擎配置如下时：

```
ENGINE MergeTree() PARTITION BY toYYYYMM(EventDate) ORDER BY (CounterID, EventDate) SETTINGS  
index_granularity=8192
```

这种情况下，这些查询：

```
SELECT count() FROM table WHERE EventDate = toDate(now()) AND CounterID = 34  
SELECT count() FROM table WHERE EventDate = toDate(now()) AND (CounterID = 34 OR CounterID = 42)  
SELECT count() FROM table WHERE ((EventDate >= toDate('2014-01-01') AND EventDate <= toDate('2014-01-31')) OR  
EventDate = toDate('2014-05-01')) AND CounterID IN (101500, 731962, 160656) AND (CounterID = 101500 OR  
EventDate != toDate('2014-05-01'))
```

ClickHouse 会依据主键索引剪掉不符合的数据，依据按月分区的分区键剪掉那些不包含符合数据的分区。

上文的查询显示，即使索引用于复杂表达式。因为读表操作是组织好的，所以，使用索引不会比完整扫描慢。

下面这个例子中，不会使用索引。

```
SELECT count() FROM table WHERE CounterID = 34 OR URL LIKE '%upyachka%'
```

要检查 ClickHouse 执行一个查询时能否使用索引，可设置 `force_index_by_date` 和 `force_primary_key`。

按月分区的分区键是只能读取包含适当范围日期的数据块。这种情况下，数据块会包含很多天（最多整月）的数据。在块中，数据按主键排序，主键第一列可能不包含日期。因此，仅使用日期而没有带主键前缀条件的查询将会导致读取超过这个日期范围。

跳数索引（分段汇总索引，实验性的）

需要设置 `allow_experimental_data_skipping_indices` 为 1 才能使用此索引。（执行 `SET allow_experimental_data_skipping_indices = 1`）。

此索引在 `CREATE` 语句的列部分里定义。

```
INDEX index_name expr TYPE type(...) GRANULARITY granularity_value
```

*MergeTree 系列的表都能指定跳数索引。

这些索引是由数据块按粒度分割后的每部分在指定表达式上汇总信息 `granularity_value` 组成（粒度大小用表引擎里 `index_granularity` 的指定）。

这些汇总信息有助于用 `where` 语句跳过大片不满足的数据，从而减少 `SELECT` 查询从磁盘读取的数据量，

示例

```
CREATE TABLE table_name  
(  
    u64 UInt64,  
    i32 Int32,  
    s String,
```

```
...  
INDEX a (u64 * i32, s) TYPE minmax GRANULARITY 3,  
INDEX b (u64 * length(s)) TYPE set(1000) GRANULARITY 4  
) ENGINE = MergeTree()  
...
```

上例中的索引能让 ClickHouse 执行下面这些查询时减少读取数据量。

```
SELECT count() FROM table WHERE s < 'z'  
SELECT count() FROM table WHERE u64 * i32 == 10 AND u64 * length(s) >= 1234
```

索引的可用类型

- **minmax**

存储指定表达式的极值（如果表达式是 `tuple`，则存储 `tuple` 中每个元素的极值），这些信息用于跳过数据块，类似主键。

- **set(max_rows)**

存储指定表达式的惟一值（不超过 `max_rows` 个，`max_rows=0` 则表示『无限制』）。这些信息可用于检查 `WHERE` 表达式是否满足某个数据块。

- **ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)**

存储包含数据块中所有 `n` 元短语的 布隆过滤器。只可用在字符串上。

可用于优化 `equals`，`like` 和 `in` 表达式的性能。

`n` - 短语长度。

`size_of_bloom_filter_in_bytes` - 布隆过滤器大小，单位字节。（因为压缩得好，可以指定比较大的值，如 256 或 512）。

`number_of_hash_functions` - 布隆过滤器中使用的 `hash` 函数的个数。

`random_seed` - `hash` 函数的随机种子。

- **tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)**

跟 `ngrambf_v1` 类似，不同于 `ngrams` 存储字符串指定长度的所有片段。它只存储被非字母数据字符分割的片段。

```
INDEX sample_index (u64 * length(s)) TYPE minmax GRANULARITY 4  
INDEX sample_index2 (u64 * length(str), i32 + f64 * 100, date, str) TYPE set(100) GRANULARITY 4  
INDEX sample_index3 (lower(str), str) TYPE ngrambf_v1(3, 256, 2, 0) GRANULARITY 4
```

并发数据访问

应对表的并发访问，我们使用多版本机制。换言之，当同时读和更新表时，数据从当前查询到的一组片段中读取。没有冗长的锁。插入不会阻碍读取。

对表的读操作是自动并行的。

列和表的 TTL

TTL 可以设置值的生命周期，它既可以为整张表设置，也可以为每个列字段单独设置。如果 `TTL` 同时作用于表和字段，ClickHouse 会使用先到期的那个。

被设置 TTL 的表，必须拥有 `日期` 或 `日期时间` 类型的字段。要定义数据的生命周期，需要在这个日期字段上使用操作符，例如：

```
TTL time_column  
TTL time_column + interval
```

要定义 `interval`，需要使用 `时间间隔` 操作符。

```
TTL date_time + INTERVAL 1 MONTH
```

```
TTL date_time + INTERVAL 15 HOUR
```

列字段 TTL

当列字段中的值过期时, ClickHouse会将它们替换成数据类型的默认值。如果分区内, 某一列的所有值均已过期, 则ClickHouse会从文件系统中删除这个分区目录下的列文件。

TTL子句不能被用于主键字段。

示例说明:

创建一张包含 **TTL** 的表

```
CREATE TABLE example_table
(
    d DateTime,
    a Int TTL d + INTERVAL 1 MONTH,
    b Int TTL d + INTERVAL 1 MONTH,
    c String
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(d)
ORDER BY d;
```

为表中已存在的列字段添加 **TTL**

```
ALTER TABLE example_table
MODIFY COLUMN
c String TTL d + INTERVAL 1 DAY;
```

修改列字段的 **TTL**

```
ALTER TABLE example_table
MODIFY COLUMN
c String TTL d + INTERVAL 1 MONTH;
```

表 TTL

当表内的数据过期时, ClickHouse会删除所有对应的行。

举例说明:

创建一张包含 **TTL** 的表

```
CREATE TABLE example_table
(
    d DateTime,
    a Int
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(d)
ORDER BY d
TTL d + INTERVAL 1 MONTH;
```

修改表的 **TTL**

```
ALTER TABLE example_table
```

```
ALTER TABLE example_table  
MODIFY TTL d + INTERVAL 1 DAY;
```

删除数据

当 ClickHouse 合并数据分区时，会删除 TTL 过期的数据。

当 ClickHouse 发现数据过期时，它将会执行一个计划外的合并。要控制这类合并的频率，你可以设置 `merge_with_ttl_timeout`。如果该值被设置的太低，它将导致执行许多的计划外合并，这可能会消耗大量资源。

如果在合并的时候执行 `SELECT` 查询，则可能会得到过期的数据。为了避免这种情况，可以在 `SELECT` 之前使用 `OPTIMIZE` 查询。

使用多个块设备进行数据存储

配置

SummingMergeTree

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

我们推荐将该引擎和 `MergeTree` 一起使用。例如，在准备做报告的时候，将完整的数据存储在 `MergeTree` 表中，并且使用 `SummingMergeTree` 来存储聚合数据。这种方法可以使你避免因为使用不正确的主键组合方式而丢失有价值的数据。

建表

```
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, ...]
```

请求参数的描述，参考 [请求描述](#)。

SummingMergeTree 的参数

- `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。

子句

创建 `SummingMergeTree` 表时，需要与创建 `MergeTree` 表时相同的子句。

► 已弃用的建表方法

用法示例

考虑如下的表：

```
CREATE TABLE summtt
(
    key UInt32,
    value UInt32
)
ENGINE = SummingMergeTree()
ORDER BY key
```

向其中插入数据：

```
:) INSERT INTO summtt Values(1,1),(1,2),(2,1)
```

ClickHouse 可能不会完整的汇总所有行（[见下文](#)），因此我们在查询中使用了聚合函数 `sum` 和 `GROUP BY` 子句。

```
SELECT key, sum(value) FROM summtt GROUP BY key
```

key	sum(value)
2	1
1	3

数据处理

当数据被插入到表中时，他们将被原样保存。ClickHouse 定期合并插入的数据片段，并在这个时候对所有具有相同主键的行中的列进行汇总，将这些行替换为包含汇总数据的一行记录。

ClickHouse 会按片段合并数据，以至于不同的数据片段中会包含具有相同主键的行，即单个汇总片段将会是不完整的。因此，聚合函数 `sum()` 和 `GROUP BY` 子句应该在（`SELECT`）查询语句中被使用，如上文中的例子所述。

汇总的通用规则

列中数值类型的值会被汇总。这些列的集合在参数 `columns` 中被定义。

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。

AggregateFunction 列中的汇总

对于 `AggregateFunction` 类型的列，ClickHouse 根据对应函数表现为 `AggregatingMergeTree` 引擎的聚合。

嵌套结构

表中可以具有以特殊方式处理的嵌套数据结构。

如果嵌套表的名称以 `Map` 结尾，并且包含至少两个符合以下条件的列：

- 第一列是数值类型 `(*Int*, Date, DateTime)`，我们称之为 `key`,
- 其他的列是可计算的 `(*Int*, Float32/64)`，我们称之为 `(values...)`,

然后这个嵌套表会被解释为一个 `key => (values...)` 的映射，当合并它们的行时，两个数据集中的元素会被根据 `key` 合并为相应的 `(values...)` 的汇总值。

示例：

```
[(1, 100)] + [(2, 150)] -> [(1, 100), (2, 150)]
[(1, 100)] + [(1, 150)] -> [(1, 250)]
```

```
[(1, 100)] + [(1, 150), (2, 150)] -> [(1, 250), (2, 150)]
[(1, 100), (2, 150)] + [(1, -100)] -> [(2, 150)]
```

请求数据时，使用 **sumMap(key,value)** 函数来对 **Map** 进行聚合。

对于嵌套数据结构，你无需在列的元组中指定列以进行汇总。

折叠树

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。更多的细节请看本文的**折叠**部分。

因此，该引擎可以显著的降低存储量并提高 **SELECT** 查询效率。

建表

```
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, ...]
```

请求参数的描述，参考[请求参数](#)。

CollapsingMergeTree 参数

- **sign** — 类型列的名称：**1** 是«状态»行，**-1** 是«取消»行。

列数据类型 — **Int8**。

子句

创建 **CollapsingMergeTree** 表时，需要与创建 **MergeTree** 表时相同的[子句](#)。

▶ 已弃用的建表方法

折叠

数据

考虑你需要为某个对象保存不断变化的数据的情景。似乎为一个对象保存一行记录并在其发生任何变化时更新记录是合乎逻辑的，但是更新操作对 DBMS 来说是昂贵且缓慢的，因为它需要重写存储中的数据。如果你需要快速的写入数据，则更新操作是不可接受的，但是你可以按下面的描述顺序地更新一个对象的变化。

在写入行的时候使用特定的列 **Sign**。如果 **Sign = 1** 则表示这一行是对象的状态，我们称之为«状态»行。如果 **Sign = -1** 则表示是对具有相同属性的状态行的取消，我们称之为«取消»行。

例如，我们想要计算用户在某个站点访问的页面页面数以及他们在那停留的时间。在某个时候，我们将用户的活动状态写入下面这样的行。

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1



一段时间后，我们写入下面的两行来记录用户活动的变化。

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	-1
4324182021466249494	6	185	1

第一行取消了这个对象（用户）的状态。它需要复制被取消的状态行的所有除了 `Sign` 的属性。

第二行包含了当前的状态。

因为我们只需要用户活动的最后状态，这些行

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1
4324182021466249494	5	146	-1

可以在折叠对象的失效（老的）状态的时候被删除。`CollapsingMergeTree` 会在合并数据片段的时候做这件事。

为什么我们每次改变需要 2 行可以阅读 **算法** 段。

这种方法的特殊属性

1. 写入的程序应该记住对象的状态从而可以取消它。«取消»字符串应该是«状态»字符串的复制，除了相反的 `Sign`。它增加了存储的初始数据的大小，但使得写入数据更快速。
2. 由于写入的负载，列中长的增长阵列会降低引擎的效率。数据越简单，效率越高。
3. `SELECT` 的结果很大程度取决于对象变更历史的一致性。在准备插入数据时要准确。在不一致的数据中会得到不可预料的结果，例如，像会话深度这种非负指标的负值。

算法

当 `ClickHouse` 合并数据片段时，每组具有相同主键的连续行被减少到不超过两行，一行 `Sign = 1`（«状态»行），另一行 `Sign = -1`（«取消»行），换句话说，数据项被折叠了。

对每个结果的数据部分 `ClickHouse` 保存：

1. 第一个«取消»和最后一个«状态»行，如果«状态»和«取消»行的数量匹配且最后一个行是«状态»行
2. 最后一个«状态»行，如果«状态»行比«取消»行多一个或一个以上。
3. 第一个«取消»行，如果«取消»行比«状态»行多一个或一个以上。
4. 没有行，在其他所有情况下。

合并会继续，但是 `ClickHouse` 会把此情况视为逻辑错误并将其记录在服务日志中。这个错误会在相同的数据被插入超过一次时出现。

因此，折叠不应该改变统计数据的结果。

变化逐渐地被折叠，因此最终几乎每个对象都只剩下了最后的状态。

`Sign` 是必须的因为合并算法不保证所有有相同主键的行都会在同一个结果数据片段中，甚至是在同一台物理服务器上。`ClickHouse` 用多线程来处理 `SELECT` 请求，所以它不能预测结果中行的顺序。如果要从 `CollapsingMergeTree` 表中获取完全«折叠»后的数据，则需要聚合。

要完成折叠，请使用 `GROUP BY` 子句和用于处理符号的聚合函数编写请求。例如，要计算数量，使用 `sum(Sign)` 而不是 `count()`。要计算某物的总和，使用 `sum(Sign * x)` 而不是 `sum(x)`，并添加 `HAVING sum(Sign) > 0` 子句。

聚合体 `count` `sum` 和 `avg` 可以用这种方式计算。如果一个对象至少有一个未被折叠的壮大，則可以计算 `min` 聚合。

外窗口的 `COUNT`, `SUM` 和 `AVG`，以及它们的组合（如 `COUNT` 加上一个窗口内的一个或多个值的总和）对于外窗口的 `MIN` 和 `MAX` 聚合无法计算，因为 `CollapsingMergeTree` 不会保存折叠状态的值的历史记录。

如果你需要在不进行聚合的情况下获取数据（例如，要检查是否存在最新值与特定条件匹配的行），你可以在 `FROM` 从句中使用 `FINAL` 修饰符。这种方法显然是更低效的。

示例

示例数据：

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1
4324182021466249494	5	146	-1
4324182021466249494	6	185	1

建表：

```
CREATE TABLE UAct
(
    UserID UInt64,
    PageViews UInt8,
    Duration UInt8,
    Sign Int8
)
ENGINE = CollapsingMergeTree(Sign)
ORDER BY UserID
```

插入数据：

```
INSERT INTO UAct VALUES (4324182021466249494, 5, 146, 1)
```

```
INSERT INTO UAct VALUES (4324182021466249494, 5, 146, -1), (4324182021466249494, 6, 185, 1)
```

我们使用两次 `INSERT` 请求来创建两个不同的数据片段。如果我们使用一个请求插入数据，ClickHouse 只会创建一个数据片段且不会执行任何合并操作。

获取数据：

```
SELECT * FROM UAct
```

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	-1
4324182021466249494	6	185	1

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1

我们看到了什么，哪里有折叠？

通过两个 `INSERT` 请求，我们创建了两个数据片段。`SELECT` 请求在两个线程中被执行，我们得到了随机顺序的行。没有发生折叠是因为还没有合并数据片段。ClickHouse 在一个我们无法预料的未知时刻合并数据片段。

因此我们需要聚合：

```
SELECT
    UserID,
    sum(PageViews * Sign) AS PageViews,
    sum(Duration * Sign) AS Duration
FROM UAct
GROUP BY UserID
HAVING sum(Sign) > 0
```

UserID	PageViews	Duration
4324182021466249494	6	185

如果我们不需要聚合并想要强制进行折叠，我们可以在 `FROM` 从句中使用 `FINAL` 修饰语。

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign
4324182021466249494	6	185	1

这种查询数据的方法是非常低效的。不要在大表中使用它。

数据副本

只有 MergeTree 系列里的表可支持副本：

- ReplicatedMergeTree
- ReplicatedSummingMergeTree
- ReplicatedReplacingMergeTree
- ReplicatedAggregatingMergeTree
- ReplicatedCollapsingMergeTree
- ReplicatedVersionedCollapsingMergetree
- ReplicatedGraphiteMergeTree

副本是表级别的，不是整个服务器级的。所以，服务器里可以同时有复制表和非复制表。

副本不依赖分片。每个分片有它自己的独立副本。

对于 `INSERT` 和 `ALTER` 语句操作数据的会在压缩的情况下被复制（更多信息，看 [ALTER](#)）。

而 `CREATE`，`DROP`，`ATTACH`，`DETACH` 和 `RENAME` 语句只会在单个服务器上执行，不会被复制。

- The `CREATE TABLE` 在运行此语句的服务器上创建一个新的可复制表。如果此表已存在其他服务器上，则给该表添加新副本。
- The `DROP TABLE` 删除运行此查询的服务器上的副本。
- The `RENAME` 重命名一个副本。换句话说，可复制表不同的副本可以有不同的名称。

要使用副本，需在配置文件中设置 ZooKeeper 集群的地址。例如：

```
<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` 和 `fallback_to_stale_replicas_for_distributed_queries` 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。

对于非常大的集群，你可以把不同的 ZooKeeper 集群用于不同的分片。然而，即使 Yandex.Metrica 集群（大约300台服务器）也证明还不需要这么做。

复制是多主异步。`INSERT` 语句（以及 `ALTER`）可以发给任意可用的服务器。数据会先插入到执行该语句的服务器上，然后被复制到其他服务器。由于它是异步的，在其他副本上最近插入的数据会有一些延迟。如果部分副本不可用，则数据在其可用时再写入。副本可用的情况下，则延迟时长是通过网络传输压缩数据块所需的时间。

默认情况下，`INSERT` 语句仅等待一个副本写入成功后返回。如果数据只成功写入一个副本后该副本所在的服务器不再存在，则存储的数据会丢失。要启用数据写入多个副本才确认返回，使用 `insert_quorum` 选项。

单个数据块写入是原子的。`INSERT` 的数据按每块最多 `max_insert_block_size = 1048576` 行进行分块，换句话说，如果 `INSERT` 插入的行少于 `1048576`，则该 `INSERT` 是原子的。

数据块会去重。对于被多次写的相同数据块（大小相同且具有相同顺序的相同行的数据块），该块仅会写入一次。这样设计的原因是万一在网络故障时客户端应用程序不知道数据是否成功写入DB，此时可以简单地重复 `INSERT`。把相同的数据发送给多个副本 `INSERT` 并不会有问题。因为这些 `INSERT` 是完全相同的（会被去重）。去重参数参看服务器设置 `merge_tree`。（注意：`Replicated*MergeTree` 才会去重，不需要 `zookeeper` 的不带 `MergeTree` 不会去重）

在复制期间，只有要插入的源数据通过网络传输。进一步的数据转换（合并）会在所有副本上以相同的方式进行处理执行。这样可以最大限度地减少网络使用，这意味着即使副本在不同的数据中心，数据同步也能工作良好。（能在不同数据中心中的同步数据是副本机制的主要目标。）

你可以给数据做任意多的副本。Yandex.Metrica 在生产中使用双副本。某些情况下，给每台服务器都使用 RAID-5 或 RAID-6 和 RAID-10。是一种相对可靠和方便的解决方案。

系统会监视副本数据同步情况，并能在发生故障后恢复。故障转移是自动的（对于小的数据差异）或半自动的（当数据差异很大时，这可能意味着有配置错误）。

创建复制表

在表引擎名称上加上 `Replicated` 前缀。例如：`ReplicatedMergeTree`。

Replicated*MergeTree 参数

- `zoo_path` — ZooKeeper 中该表的路径。
- `replica_name` — ZooKeeper 中的该表的副本名称。

示例：

```
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)
```

已弃用的建表语法示例：

```
CREATE TABLE table_name
(
    EventDate DateTime,
    CounterID UInt32,
    UserID UInt32
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/table_name', '{replica}', EventDate,
intHash32(UserID), (CounterID, EventDate, intHash32(UserID), EventTime), 8192)
```

如上例所示，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。

在每个副本服务器上运行 CREATE TABLE 查询。将创建新的复制表，或给现有表添加新副本。

如果其他副本上已包含了某些数据，在表上添加新副本，则在运行语句后，数据会从其他副本复制到新副本。换句话说，新副本会与其他副本同步。

要删除副本，使用 DROP TABLE。但它只删除那个 - 位于运行该语句的服务器上的副本。

如果 ZooKeeper 不可用

如果服务器启动时 ZooKeeper 不可用，则复制表会切换为只读模式。系统会定期尝试去连接 ZooKeeper。

如果在 `INSERT` 期间 ZooKeeper 不可用，或者在与 ZooKeeper 交互时发生错误，则抛出异常。

连接到 ZooKeeper 后，系统会检查本地文件系统中的数据集是否与预期的数据集（ZooKeeper 存储此信息）一致。如果存在轻微的不一致，系统会通过与副本同步数据来解决。

如果系统检测到损坏的数据片段（文件大小错误）或无法识别的片段（写入文件系统但未记录在 ZooKeeper 中的部分），则会把它们移动到 ‘detached’ 子目录（不会删除）。而副本中其他任何缺少的但正常数据片段都会被复制同步。

注意，ClickHouse 不会执行任何破坏性操作，例如自动删除大量数据。

当服务器启动（或与 ZooKeeper 建立新会话）时，它只检查所有文件的数量和大小。如果文件大小一致但中间某处已有字节被修改过，不会立即被检测到，只有在尝试读取 `SELECT` 查询的数据时才会检测到。该查询会引发校验和不匹配或压缩块大小不一致的异常。这种情况下，数据片段会添加到验证队列中，并在必要时从其他副本中复制。

如果本地数据集与预期数据的差异太大，则会触发安全机制。服务器在日志中记录此内容并拒绝启动。这种情况很可能是配置错误，例如，一个分片上的副本意外配置为别的分片上的副本。然而，此机制的阈值设置得相当低，在正常故障恢复期间可能会出现这种情况。在这种情况下，数据恢复则是半自动模式，通过用户主动操作触发。

要触发启动恢复，可在 ZooKeeper 中创建节点 `/path_to_table/replica_name flags/force_restore_data`，节点值可以是任何内容，或运行命令来恢复所有的可复制表：

```
sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data
```

然后重启服务器。启动时，服务器会删除这些标志并开始恢复。

在数据完全丢失后的恢复

如果其中一个服务器的所有数据和元数据都消失了，请按照以下步骤进行恢复：

1. 在服务器上安装 ClickHouse。在包含分片标识符和副本的配置文件中正确定义宏配置，如果有用到的话，
2. 如果服务器上有非复制表则必须手动复制，可以从副本服务器上（在 `/var/lib/clickhouse/data/db_name/table_name/` 目录中）复制它们的数据。
3. 从副本服务器上中复制位于 `/var/lib/clickhouse/metadata/` 中的表定义信息。如果在表定义信息中显式指定了分片或副本标识符，请更正它以使其对应于该副本。（另外，启动服务器，然后会在 `/var/lib/clickhouse/metadata/` 中的 `.sql` 文件中生成所有的 `ATTACH TABLE` 语句。）
4. 要开始恢复，ZooKeeper 中创建节点 `/path_to_table/replica_name flags/force_restore_data`，节点内容不限，或运行命令来恢复所有复制的表：`sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data`

然后启动服务器（如果它已运行则重启）。数据会从副本中下载。

另一种恢复方式是从 ZooKeeper (`/path_to_table/replica_name`) 中删除有数据丢的副本的所有元信息，然后再按照《[创建可复制表](#)》中的描述重新创建副本。

恢复期间的网络带宽没有限制。特别注意这一点，尤其是要一次恢复很多副本。

MergeTree 转换为 ReplicatedMergeTree

我们使用 `MergeTree` 来表示 `MergeTree` 系列 中的所有表引擎，`ReplicatedMergeTree` 同理。

如果你有一个手动同步的 `MergeTree` 表，您可以将其转换为可复制表。如果你已经在 `MergeTree` 表中收集了大量数据，并且现在要启用复制，则可以执行这些操作。

如果各个副本上的数据不一致，则首先对其进行同步，或者除保留的一个副本外，删除其他所有副本上的数据。

重命名现有的 `MergeTree` 表，然后使用旧名称创建 `ReplicatedMergeTree` 表。

将数据从旧表移动到新表 (`/var/lib/clickhouse/data/db_name/table_name/`) 目录内的 ‘detached’ 目录中。
然后在其中一个副本上运行 `ALTER TABLE ATTACH PARTITION`，将这些数据片段添加到工作集中。

ReplicatedMergeTree 转换为 MergeTree

使用其他名称创建 MergeTree 表。将具有 ReplicatedMergeTree 表数据的目录中的所有数据移动到新表的数据目录中。然后删除 ReplicatedMergeTree 表并重新启动服务器。

如果你想在不启动服务器的情况下清除 ReplicatedMergeTree 表：

- 删除元数据目录中的相应 .sql 文件 (`/var/lib/clickhouse/metadata/`) 。
- 删除 ZooKeeper 中的相应路径 (`/path_to_table/replica_name`) 。

之后，你可以启动服务器，创建一个 MergeTree 表，将数据移动到其目录，然后重新启动服务器。

当 ZooKeeper 集群中的元数据丢失或损坏时恢复方法

如果 ZooKeeper 中的数据丢失或损坏，如上所述，你可以通过将数据转移到非复制表来保存数据。

更换麦树

该引擎和 MergeTree 的不同之处在于它会删除具有相同主键的重复项。

数据的去重只会在合并的过程中出现。合并在未知的时间在后台进行，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，但请不要指望使用它，因为 `OPTIMIZE` 语句会引发对大量数据的读和写。

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

```
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, ...]
```

请求参数的描述，参考[请求参数](#)。

替换树参数

- `ver` — 版本列。类型为 `UInt*`, `Date` 或 `DateTime`。可选参数。

合并的时候，`ReplacingMergeTree` 从所有具有相同主键的行中选择一行留下：

- 如果 `ver` 列未指定，选择最后一条。
- 如果 `ver` 列已指定，选择 `ver` 值最大的版本。

子句

创建 `ReplacingMergeTree` 表时，需要与创建 `MergeTree` 表时相同的[子句](#)。

► 已弃用的建表方法

自定义分区键

MergeTree 系列的表（包括 可复制表）可以使用分区。基于 MergeTree 表的 物化视图 也支持分区。

一个分区是指按指定规则逻辑组合一起的表的记录集。可以按任意标准进行分区，如按月，按日或按事件类型。为了减少需要操作的数据，每个分区都是分开存储的。访问数据时，ClickHouse 尽量使用这些分区的最小子集。

分区是在 [建表](#) 的 `PARTITION BY expr` 子句中指定。分区键可以是关于列的任何表达式。例如，指定按月分区，表达式为 `toYYYYMM(date_column)`：

```
CREATE TABLE visits
(
    VisitDate Date,
    Hour UInt8,
    ClientID UUID
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(VisitDate)
ORDER BY Hour;
```

分区键也可以是表达式元组（类似 [主键](#)）。例如：

```
ENGINE = ReplicatedCollapsingMergeTree('/clickhouse/tables/name', 'replica1', Sign)
PARTITION BY (toMonday(StartDate), EventType)
ORDER BY (CounterID, StartDate, intHash32(UserID));
```

上例中，我们设置按一周内的事件类型分区。

新数据插入到表中时，这些数据会存储为按主键排序的新片段（块）。插入后 10-15 分钟，同一分区的各个片段会合并为一个整个片段。

注意

那些有相同分区表达式值的数据片段才会合并。这意味着 [你不应该用太精细的分区方案](#)（超过一千个分区）。否则，会因为文件系统中的文件数量和需要找开的文件描述符过多，导致 `SELECT` 查询效率不佳。

可以通过 [系统。零件](#) 表查看表片段和分区信息。例如，假设我们有一个 `visits` 表，按月分区。对 `system.parts` 表执行 `SELECT`：

```
SELECT
    partition,
    name,
    active
FROM system.parts
WHERE table = 'visits'
```

partition	name	active
201901	201901_1_3_1	0
201901	201901_1_9_2	1
201901	201901_8_8_0	0
201901	201901_9_9_0	0
201902	201902_4_6_1	1
201902	201902_10_10_0	1
201902	201902_11_11_0	1

`partition` 列存储分区的名称。此示例中有两个分区：`201901` 和 `201902`。在 [ALTER ... PARTITION](#) 语句中你可以使用该列值来指定分区名称。

`name` 列为分区中数据片段的名称。在 **ALTER ATTACH PART** 语句中你可以使用此列值中来指定片段名称。

这里我们拆解下第一部分的名称：`201901_1_3_1`：

- `201901` 是分区名称。
- `1` 是数据块的最小编号。
- `3` 是数据块的最大编号。
- `1` 是块级别（即在由块组成的合并树中，该块在树中的深度）。

注意

旧类型表的片段名称为：`20190117_20190123_2_2_0`（最小日期 - 最大日期 - 最小块编号 - 最大块编号 - 块级别）。

`active` 列为片段状态。`1` 激活状态；`0` 非激活状态。非激活片段是那些在合并到较大片段之后剩余的源数据片段。损坏的数据片段也表示为非活动状态。

正如在示例中所看到的，同一分区中有几个独立的片段（例如，`201901_1_3_1` 和 `201901_1_9_2`）。这意味着这些片段尚未合并。ClickHouse 大约在插入后 15 分钟定期报告合并操作，合并插入的数据片段。此外，你也可以使用 **OPTIMIZE** 语句直接执行合并。例：

```
OPTIMIZE TABLE visits PARTITION 201902;
```

partition	name	active
201901	201901_1_3_1	0
201901	201901_1_9_2	1
201901	201901_8_8_0	0
201901	201901_9_9_0	0
201902	201902_4_6_1	0
201902	201902_4_11_2	1
201902	201902_10_10_0	0
201902	201902_11_11_0	0

非激活片段会在合并后的 10 分钟左右删除。

查看片段和分区信息的另一种方法是进入表的目录：`/var/lib/clickhouse/data/<database>/<table>/`。例如：

```
dev:/var/lib/clickhouse/data/default/visits$ ls -l
total 40
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 1 16:48 201901_1_3_1
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 16:17 201901_1_9_2
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 15:52 201901_8_8_0
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 15:52 201901_9_9_0
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 16:17 201902_10_10_0
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 16:17 201902_11_11_0
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 16:19 201902_4_11_2
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 5 12:09 201902_4_6_1
drwxr-xr-x 2 clickhouse clickhouse 4096 Feb 1 16:48 detached
```

文件夹 '`201901_1_1_0`'，'`201901_1_7_1`' 等是片段的目录。每个片段都与一个对应的分区相关，并且只包含这个月的数据（本例中的表按月分区）。

`detached` 目录存放着使用 **DETACH** 语句从表中分离的片段。损坏的片段也会移到该目录，而不是删除。服务器不使用 `detached` 目录中的片段。可以随时添加，删除或修改此目录中的数据 - 在运行 **ATTACH** 语句前，服务器不会感知到。

注意，操作服务器时，你不能干预其他系统上的片段或其数据，因为服务器不会感知到这些修改。对于非复制表，可以在服务器停止时执行这些操作，但不建议这样做。对于复制表，在任何情况下都不要更改片段文件。

ClickHouse 支持对分区执行这些操作：删除分区，从一个表复制到另一个表，或创建备份。了解分区的所有操作，请参阅 [分区和片段的操作](#) 一节。

StripeLog

该引擎属于日志引擎系列。请在 [日志引擎系列](#) 文章中查看引擎的共同属性和差异。

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。

建表

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    column1_name [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    column2_name [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = StripeLog
```

查看 [建表](#) 请求的详细说明。

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。使用 `ORDER BY` 子句对行进行排序。

使用示例

建表：

```
CREATE TABLE stripe_log_table
(
    timestamp DateTime,
    message_type String,
    message String
)
ENGINE = StripeLog
```

插入数据：

```
INSERT INTO stripe_log_table VALUES (now(),'REGULAR','The first regular message')
INSERT INTO stripe_log_table VALUES (now(),'REGULAR','The second regular message'),(now(),'WARNING','The first warning message')
```

我们使用两次 `INSERT` 请求从而在 `data.bin` 文件中创建两个数据块。

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。例如：

```
SELECT * FROM stripe_log_table
```

timestamp	message_type	message
2019-01-18 14:27:32	REGULAR	The second regular message
2019-01-18 14:34:53	WARNING	The first warning message

timestamp	message_type	message
2019-01-18 14:23:43	REGULAR	The first regular message

对结果排序（默认增序）：

```
SELECT * FROM stripe_log_table ORDER BY timestamp
```

timestamp	message_type	message
2019-01-18 14:23:43	REGULAR	The first regular message
2019-01-18 14:27:32	REGULAR	The second regular message
2019-01-18 14:34:53	WARNING	The first warning message

TinyLog

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

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。

日志

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。

日志引擎系列

这些引擎是为了需要写入许多小数据量（少于一百万行）的表的场景而开发的。

这系列的引擎有：

- [StripeLog](#)
- [日志](#)

- [TinyLog](#)

共同属性

引擎：

- 数据存储在磁盘上。
- 写入时将数据追加在文件末尾。
- 不支持[突变](#)操作。
- 不支持索引。

这意味着`SELECT`在范围查询时效率不高。

- 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。

差异

[Log](#) 和 [StripeLog](#) 引擎支持：

- 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。

- 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。

[TingLog](#) 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。[TingLog](#) 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。

JDBC

允许ClickHouse通过以下方式连接到外部数据库 [JDBC](#).

要实现JDBC连接，ClickHouse使用单独的程序 [ツ暗エツ汎环催ツ団ツ法ツ人](#) 这应该作为守护进程运行。

该引擎支持[可为空](#) 数据类型。

创建表

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
    columns list...
```

```
)  
ENGINE = JDBC(dbms_uri, external_database, external_table)
```

发动机参数

- dbms_uri — URI of an external DBMS.

格式: jdbc:<driver_name>://<host_name>:<port>/?user=<username>&password=<password>.

Mysql的示例: jdbc:mysql://localhost:3306/?user=root&password=root.

- external_database — Database in an external DBMS.
- external_table — Name of the table in external_database.

用法示例

通过直接与它的控制台客户端连接在MySQL服务器中创建一个表:

```
mysql> CREATE TABLE `test`.`test` (  
-> `int_id` INT NOT NULL AUTO_INCREMENT,  
-> `int_nullable` INT NULL DEFAULT NULL,  
-> `float` FLOAT NOT NULL,  
-> `float_nullable` FLOAT NULL DEFAULT NULL,  
-> PRIMARY KEY (`int_id`);  
Query OK, 0 rows affected (0,09 sec)
```

```
mysql> insert into test (`int_id`, `float`) VALUES (1,2);  
Query OK, 1 row affected (0,00 sec)
```

```
mysql> select * from test;  
+-----+-----+-----+  
| int_id | int_nullable | float | float_nullable |  
+-----+-----+-----+  
| 1 | NULL | 2 | NULL |  
+-----+-----+-----+  
1 row in set (0,00 sec)
```

在ClickHouse服务器中创建表并从中选择数据:

```
CREATE TABLE jdbc_table  
(  
`int_id` Int32,  
`int_nullable` Nullable(Int32),  
`float` Float32,  
`float_nullable` Nullable(Float32)  
)  
ENGINE JDBC('jdbc:mysql://localhost:3306/?user=root&password=root', 'test', 'test')
```

```
SELECT *  
FROM jdbc_table
```

int_id	int_nullable	float	float_nullable
1	NULL	2	NULL

另请参阅

- JDBC表函数.

ODBC

允许ClickHouse通过以下方式连接到外部数据库 ODBC.

为了安全地实现ODBC连接，ClickHouse使用单独的程序 `clickhouse-odbc-bridge`. 如果直接从ODBC驱动程序加载 `clickhouse-server`，驱动程序问题可能会导致ClickHouse服务器崩溃。ClickHouse自动启动 `clickhouse-odbc-bridge` 当它是必需的。ODBC桥程序是从相同的软件包作为安装 `clickhouse-server`.

该引擎支持 可为空 数据类型。

创建表

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1],
    name2 [type2],
    ...
)
ENGINE = ODBC(connection_settings, external_database, external_table)
```

请参阅的详细说明 [CREATE TABLE](#) 查询。

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 投 ClickHouse数据类型的值。

发动机参数

- `connection_settings` — Name of the section with connection settings in the `odbc.ini` 文件
- `external_database` — Name of a database in an external DBMS.
- `external_table` — Name of a table in the `external_database`.

用法示例

通过ODBC从本地MySQL安装中检索数据

此示例检查Ubuntu Linux18.04和MySQL服务器5.7。

确保安装了unixODBC和MySQL连接器。

默认情况下（如果从软件包安装），ClickHouse以用户身份启动 `clickhouse`. 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;
```

然后配置连接 `/etc/odbc.ini`.

```
$ cat /etc/odbc.ini
[mysqlconn]
```

```
DRIVER = /usr/local/lib/libmyodbc5w.so
SERVER = 127.0.0.1
PORT = 3306
DATABASE = test
USERNAME = clickhouse
PASSWORD = clickhouse
```

您可以使用 `isql unixodbc` 安装中的实用程序。

```
$ isql -v mysqlconn
+-----+
| Connected!
| |
...|
```

MySQL 中的表：

```
mysql> CREATE TABLE `test`.`test` (
->   `int_id` INT NOT NULL AUTO_INCREMENT,
->   `int_nullable` INT NULL DEFAULT NULL,
->   `float` FLOAT NOT NULL,
->   `float_nullable` FLOAT NULL DEFAULT NULL,
->   PRIMARY KEY (`int_id`));
Query OK, 0 rows affected (0,09 sec)

mysql> insert into test (`int_id`, `float`) VALUES (1,2);
Query OK, 1 row affected (0,00 sec)

mysql> select * from test;
+-----+-----+-----+
| int_id | int_nullable | float | float_nullable |
+-----+-----+-----+
|    1 |      NULL |    2 |      NULL |
+-----+-----+-----+
1 row in set (0,00 sec)
```

ClickHouse 中的表，从 MySQL 表中检索数据：

```
CREATE TABLE odbc_t
(
  `int_id` Int32,
  `float_nullable` Nullable(Float32)
)
ENGINE = ODBC('DSN=mysqlconn', 'test', 'test')
```

```
SELECT * FROM odbc_t
```

int_id	float_nullable
1	NULL

另请参阅

- [ODBC 外部字典](#)
- [ODBC 表函数](#)

HDFS

该引擎提供了集成 [Apache Hadoop](#) 生态系统通过允许管理数据 [HDFS](#) 通过 ClickHouse。这个引擎是相似的到 [文件](#) 和 [URL](#) 引擎，但提供 Hadoop 特定的功能。

用途

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

该 `URI` 参数是 HDFS 中的整个文件 `URI`。

该 `format` 参数指定一种可用的文件格式。执行

`SELECT` 查询时，格式必须支持输入，并执行

`INSERT` queries – for output. The available formats are listed in the [格式](#) 科。

路径部分 `URI` 可能包含水珠。在这种情况下，表将是只读的。

示例：

1. 设置 `hdfs_engine_table` 表：

```
CREATE TABLE hdfs_engine_table (name String, value UInt32) ENGINE=HDFS('hdfs://hdfs1:9000/other_storage', 'TSV')
```

2. 填充文件：

```
INSERT INTO hdfs_engine_table VALUES ('one', 1), ('two', 2), ('three', 3)
```

3. 查询数据：

```
SELECT * FROM hdfs_engine_table LIMIT 2
```

name	value
one	1
two	2

实施细节

- 读取和写入可以并行

- 不支持：

- `ALTER` 和 `SELECT...SAMPLE` 操作。
- 索引。
- 复制。

路径中的水珠

多个路径组件可以具有 `glob`s。对于正在处理的文件应该存在并匹配到整个路径模式。文件列表确定在 `SELECT`（不在 `CREATE` 时刻）。

- `*` — Substitutes any number of any characters except / 包括空字符串。
- `?` — Substitutes any single character.
- `{some_string,another_string,yet_another_one}` — Substitutes any of strings 'some_string', 'another_string',

- 'yet_another_one'.
- `{N..M}` — Substitutes any number in range from N to M including both borders.

建筑与 `{}` 类似于 [远程](#) 表功能。

示例

1. 假设我们在HDFS上有几个TSV格式的文件，其中包含以下Uri:

- 'hdfs://hdfs1:9000/some_dir/some_file_1'
- 'hdfs://hdfs1:9000/some_dir/some_file_2'
- 'hdfs://hdfs1:9000/some_dir/some_file_3'
- 'hdfs://hdfs1:9000/another_dir/some_file_1'
- 'hdfs://hdfs1:9000/another_dir/some_file_2'
- 'hdfs://hdfs1:9000/another_dir/some_file_3'

1. 有几种方法可以创建由所有六个文件组成的表:

```
CREATE TABLE table_with_range (name String, value UInt32) ENGINE =  
HDFS('hdfs://hdfs1:9000/{some,another}_dir/some_file_{1..3}', 'TSV')
```

另一种方式:

```
CREATE TABLE table_with_question_mark (name String, value UInt32) ENGINE =  
HDFS('hdfs://hdfs1:9000/{some,another}_dir/some_file_?', 'TSV')
```

表由两个目录中的所有文件组成（所有文件都应满足query中描述的格式和模式）:

```
CREATE TABLE table_with_asterisk (name String, value UInt32) ENGINE =  
HDFS('hdfs://hdfs1:9000/{some,another}_dir/*', 'TSV')
```

警告

如果文件列表包含带有前导零的数字范围，请单独使用带有大括号的构造或使用 `?`。

示例

创建具有名为文件的表 `file000`, `file001`, ..., `file999`:

```
CREARE TABLE big_table (name String, value UInt32) ENGINE = HDFS('hdfs://hdfs1:9000/big_dir/file{0..9}{0..9}{0..9}',  
'CSV')
```

虚拟列

- `_path` — Path to the file.
- `_file` — Name of the file.

另请参阅

- [虚拟列](#)

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。

调用格式：

```
MySQL('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause']);
```

调用参数

- `host:port` — MySQL 服务器地址。
- `database` — 数据库的名称。
- `table` — 表名称。
- `user` — 数据库用户。
- `password` — 用户密码。
- `replace_query` — 将 `INSERT INTO` 查询是否替换为 `REPLACE INTO` 的标志。如果 `replace_query=1`，则替换查询。
- `'on_duplicate_clause'` — 将 `ON DUPLICATE KEY UPDATE 'on_duplicate_clause'` 表达式添加到 `INSERT` 查询语句中。例如：`impression = VALUES(impression) + impression`。如果需要指定 `'on_duplicate_clause'`，则需要设置 `replace_query=0`。如果同时设置 `replace_query = 1` 和 `'on_duplicate_clause'`，则会抛出异常。

此时，简单的 `WHERE` 子句（例如 `=, !=, >, >=, <, <=`）是在 MySQL 服务器上执行。

其余条件以及 `LIMIT` 采样约束语句仅在对 MySQL 的查询完成后才在 ClickHouse 中执行。

MySQL 引擎不支持 可为空 数据类型，因此，当从 MySQL 表中读取数据时，`NULL` 将转换为指定列类型的默认值（通常为 0 或空字符串）。

卡夫卡

此引擎与 [Apache Kafka](#) 结合使用。

Kafka 特性：

- 发布或者订阅数据流。
- 容错存储机制。
- 处理流数据。

老版格式：

```
Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format  
[, kafka_row_delimiter, kafka_schema, kafka_num_consumers])
```

新版格式：

```
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
```

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** 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。

示例：

```
CREATE TABLE queue (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');

SELECT * FROM queue LIMIT 5;

CREATE TABLE queue2 (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka SETTINGS kafka_broker_list = 'localhost:9092',
    kafka_topic_list = 'topic',
    kafka_group_name = 'group1',
    kafka_format = 'JSONEachRow',
    kafka_num_consumers = 4;

CREATE TABLE queue2 (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1')
    SETTINGS kafka_format = 'JSONEachRow',
    kafka_num_consumers = 4;
```

消费的消息会被自动追踪，因此每个消息在不同的消费组里只会记录一次。如果希望获得两次数据，则使用另一个组名创建副本。

消费组可以灵活配置并且在集群之间同步。例如，如果群集中有10个主题和5个表副本，则每个副本将获得2个主题。如果副本数量发生变化，主题将自动在副本中重新分配。了解更多信息请访问 <http://kafka.apache.org/intro>。

`SELECT` 查询对于读取消息并不是很有用（调试除外），因为每条消息只能被读取一次。使用物化视图创建实时线程更实用。您可以这样做：

1. 使用引擎创建一个 `Kafka` 消费者并作为一条数据流。
2. 创建一个结构表。
3. 创建物化视图，改视图会在后台转换引擎中的数据并将其放入之前创建的表中。

当 `MATERIALIZED VIEW` 添加至引擎，它将会在后台收集数据。可以持续不断地从 `Kafka` 收集数据并通过 `SELECT` 将数据转换为所需要的格式。

示例：

```
CREATE TABLE queue (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');
```

```
CREATE TABLE daily (
    day Date,
    level String,
    total UInt64
) ENGINE = SummingMergeTree(day, (day, level), 8192);

CREATE MATERIALIZED VIEW consumer TO daily
AS SELECT toDate(toDateTime(timestamp)) AS day, level, count() as total
FROM queue GROUP BY day, level;

SELECT level, sum(total) FROM daily GROUP BY level;
```

为了提高性能，接受的消息被分组为 `max_insert_block_size` 大小的块。如果未在 `stream_flush_interval_ms` 毫秒内形成块，则不关心块的完整性，都会将数据刷新到表中。

停止接收主题数据或更改转换逻辑，请 `detach` 物化视图：

```
DETACH TABLE consumer;
ATTACH TABLE consumer;
```

如果使用 `ALTER` 更改目标表，为了避免目标表与视图中的数据之间存在差异，推荐停止物化视图。

配置

与 `GraphiteMergeTree` 类似，`Kafka` 引擎支持使用 ClickHouse 配置文件进行扩展配置。可以使用两个配置键：全局 (`kafka`) 和 主题级别 (`kafka_*`)。首先应用全局配置，然后应用主题级配置（如果存在）。

```
<!-- Global configuration options for all tables of Kafka engine type -->
<kafka>
    <debug>cgrp</debug>
    <auto_offset_reset>smallest</auto_offset_reset>
</kafka>

<!-- Configuration specific for topic "logs" -->
<kafka_logs>
    <retry_backoff_ms>250</retry_backoff_ms>
    <fetch_min_bytes>100000</fetch_min_bytes>
</kafka_logs>
```

有关详细配置选项列表，请参阅 [librdkafka 配置参考](#)。在 ClickHouse 配置中使用下划线 (`_`)，并不是使用点 (`.`)。例如，`check.crcs=true` 将是 `<check_crcs>true</check_crcs>`。

Generaterandom

`GenerateRandom` 表引擎为给定的表架构生成随机数据。

使用示例：

- 在测试中使用填充可重复的大表。
- 为模糊测试生成随机输入。

在 ClickHouse 服务器中的使用

```
ENGINE = GenerateRandom(random_seed, max_string_length, max_array_length)
```

该 `max_array_length` 和 `max_string_length` 参数指定所有的最大长度
数组列和字符串相应地在生成的数据中。

生成表引擎仅支持 `SELECT` 查询。

它支持所有 [数据类型](#) 可以存储在一个表中，除了 `LowCardinality` 和 `AggregateFunction`.

示例：

1. 设置 `generate_engine_table` 表：

```
CREATE TABLE generate_engine_table (name String, value UInt32) ENGINE = GenerateRandom(1, 5, 3)
```

2. 查询数据：

```
SELECT * FROM generate_engine_table LIMIT 3
```

name	value
c4xJ	1412771199
r	1791099446
7#\$	124312908

实施细节

- 不支持：
 - `ALTER`
 - `SELECT ... SAMPLE`
 - `INSERT`
 - 指数
 - 复制

MaterializedView

物化视图的使用（更多信息请参阅 [CREATE TABLE](#)）。它需要使用一个不同的引擎来存储数据，这个引擎要在创建物化视图时指定。当从表中读取时，它就会使用该引擎。

Null

当写入 `Null` 类型的表时，将忽略数据。从 `Null` 类型的表中读取时，返回空。

但是，可以在 `Null` 类型的表上创建物化视图。写入表的数据将转发到视图中。

URL(URL, 格式)

用于管理远程 HTTP/HTTPS 服务器上的数据。该引擎类似 [文件](#) 引擎。

在 ClickHouse 服务器中使用引擎

`Format` 必须是 ClickHouse 可以用于 `SELECT` 查询的一种格式，若有必要，还要可用于 `INSERT`。有关支持格式的完整列表，请查看 [格式](#)。

`URL` 必须符合统一资源定位符的结构。指定的 `URL` 必须指向一个

HTTP 或 HTTPS 资源，手工输入网址。

HTTP 或 HTTPS 服务器。对于服务端响应，
不需要任何额外的 HTTP 头标记。

INSERT 和 SELECT 查询会分别转换为 POST 和 GET 请求。

对于 POST 请求的处理，远程服务器必须支持
分块传输编码。

示例：

1. 在 Clickhouse 服务上创建一个 url_engine_table 表：

```
CREATE TABLE url_engine_table (word String, value UInt64)
ENGINE=URL('http://127.0.0.1:12345/', CSV)
```

2. 用标准的 Python 3 工具库创建一个基本的 HTTP 服务并
启动它：

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class CSVHTTPServer(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/csv')
        self.end_headers()

        self.wfile.write(bytes('Hello,1\nWorld,2\n', "utf-8"))

    if __name__ == "__main__":
        server_address = ('127.0.0.1', 12345)
        HTTPServer(server_address, CSVHTTPServer).serve_forever()
```

```
python3 server.py
```

3. 查询请求：

```
SELECT * FROM url_engine_table
```

word	value
Hello	1
World	2

功能实现

- 读写操作都支持并发
- 不支持：
 - ALTER 和 SELECT...SAMPLE 操作。
 - 索引。
 - 副本。

分布

分布式引擎本身不存储数据，但可以在多个服务器上进行分布式查询。

读是自动并行的。读取时，远程服务器表的索引（如果有的话）会被使用。

分布式引擎参数：服务器配置文件中的集群名，远程数据库名，远程表名，数据分片键（可选）。

示例：

```
Distributed(logs, default, hits[, sharding_key])
```

将会从位于«logs»集群中 default.hits 表所有服务器上读取数据。

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。

数据库名参数除了用数据库名之外，也可用返回字符串的常量表达式。例如：currentDatabase()。

logs - 服务器配置文件中的集群名称。

集群示例配置如下：

```
<remote_servers>
  <logs>
    <shard>
      <!-- Optional. Shard weight when writing data. Default: 1. -->
      <weight>1</weight>
      <!-- Optional. Whether to write data to just one of the replicas. Default: false (write data to all replicas). -->
      <internal_replication>false</internal_replication>
      <replica>
        <host>example01-01-1</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>example01-01-2</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <weight>2</weight>
      <internal_replication>false</internal_replication>
      <replica>
        <host>example01-02-1</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>example01-02-2</host>
        <secure>1</secure>
        <port>9440</port>
      </replica>
    </shard>
  </logs>
</remote_servers>
```

这里定义了一个名为'logs'的集群，它由两个分片组成，每个分片包含两个副本。

分片是指包含数据不同部分的服务器（要读取所有数据，必须访问所有分片）。

副本是存储复制数据的服务器（要读取所有数据，访问任一副本上的数据即可）。

集群名称不能包含点号。

每个服务器需要指定 host，port，和可选的 user，password，secure，compression 的参数：

- host - 远程服务器地址。可以域名、IPv4或IPv6。如果指定域名，则服务在启动时发起一个 DNS 请求，并且请求结果会在服务器运行期间一直被记录。如果 DNS 请求失败，则服务不会启动。如果你修改了 DNS 记录，则需要重启服务。
- port - 消息传递的 TCP 端口（「tcp_port」配置通常设为 9000）。不要跟 http_port 混淆。
- user - 用于连接远程服务器的用户名。默认值：default。该用户必须有权限访问该远程服务器。访问权限配置在 user.xml 文件中。更多信息，请参见“访问权限”部分。

- `password` - 用于连接远程服务器的密码。默认值：空字符串。
- `secure` - 是否使用SSL进行连接，设为`true`时，通常也应该设置 `port = 9440`。服务器也要监听 `<tcp_port_secure>9440</tcp_port_secure>` 并有正确的证书。
- `compression` - 是否使用数据压缩。默认值：`true`。

配置了副本，读取操作会从每个分片里选择一个可用的副本。可配置负载平衡算法（挑选副本的方式） - 请参阅《load_balancing》设置。

如果跟服务器的连接不可用，则在尝试短超时的重连。如果重连失败，则选择下一个副本，依此类推。如果跟所有副本的连接尝试都失败，则尝试用相同的方式再重复几次。

该机制有利于系统可用性，但不保证完全容错：如有远程服务器能够接受连接，但无法正常工作或状况不佳。

你可以配置一个（这种情况下，查询操作更应该称为远程查询，而不是分布式查询）或任意多个分片。在每个分片中，可以配置一个或任意多个副本。不同分片可配置不同数量的副本。

可以在配置中配置任意数量的集群。

要查看集群，可使用《system.clusters》表。

通过分布式引擎可以像使用本地服务器一样使用集群。但是，集群不是自动扩展的：你必须编写集群配置到服务器配置文件中（最好，给所有集群的服务器写上完整配置）。

不支持用分布式表查询别的分布式表（除非该表只有一个分片）。或者说，要用分布表查查询《最终》的数据表。

分布式引擎需要将集群信息写入配置文件。配置文件中的集群信息会即时更新，无需重启服务器。如果你每次是要向不确定的一组分片和副本发送查询，则不适合创建分布式表 - 而应该使用《远程》表函数。请参阅《表函数》部分。

向集群写数据的方法有两种：

一，自己指定要将哪些数据写入哪些服务器，并直接在每个分片上执行写入。换句话说，在分布式表上《查询》，在数据表上 `INSERT`。

这是最灵活的解决方案 - 你可以使用任何分片方案，对于复杂业务特性的需求，这可能是非常重要的。

这也是最佳解决方案，因为数据可以完全独立地写入不同的分片。

二，在分布式表上执行 `INSERT`。在这种情况下，分布式表会跨服务器分发插入数据。

为了写入分布式表，必须要配置分片键（最后一个参数）。当然，如果只有一个分片，则写操作在没有分片键的情况下也能工作，因为这种情况下分片键没有意义。

每个分片都可以在配置文件中定义权重。默认情况下，权重等于1。数据依据分片权重按比例分发到分片上。例如，如果有两个分片，第一个分片的权重是9，而第二个分片的权重是10，则发送 `9 / 19` 的行到第一个分片，`10 / 19` 的行到第二个分片。

分片可在配置文件中定义 ‘internal_replication’ 参数。

此参数设置为《true》时，写操作只选一个正常的副本写入数据。如果分布式表的子表是复制表(*ReplicaMergeTree)，请使用此方案。换句话说，这其实是把数据的复制工作交给实际需要写入数据的表本身而不是分布式表。

若此参数设置为《false》（默认值），写操作会将数据写入所有副本。实质上，这意味着要分布式表本身来复制数据。这种方式不如使用复制表的好，因为不会检查副本的一致性，并且随着时间的推移，副本数据可能会有些不一样。

选择将一行数据发送到哪个分片的方法是，首先计算分片表达式，然后将这个计算结果除以所有分片的权重总和得到余数。该行会发送到那个包含该余数的从‘prev_weight’到‘prev_weights + weight’的半闭半开区间对应的分片上，其中‘prev_weights’是该分片前面的所有分片的权重和，‘weight’是该分片的权重。例如，如果有两个分片，第一个分片权重为9，而第二个分片权重为10，则余数在 [0,9) 中的行发给第一个分片，余数在 [9,19) 中的行发给第二个分片。

分片表达式可以是由常量和表列组成的任何返回整数表达式。例如，您可以使用表达式 ‘rand()’ 来随机分配数据，或者使用 ‘UserID’ 来按用户 ID 的余数分布（相同用户的数据将分配到单个分片上，这可降低带有用户信息的 IN 和 JOIN 的语句运行的复杂度）。如果该列数据分布不够均匀，可以将其包装在散列函数中：`intHash64(UserID)`。

这种简单的用余数来选择分片的方案是有局限的，并不总适用。它适用于中型和大型数据（数十台服务器）的场景，但不适用于巨量数据（数百台或更多服务器）的场景。后一种情况下，应根据业务特性需求考虑的分片方案，而不是直接用分布式表的

多分片。

SELECT 查询会被发送到所有分片，并且无论数据在分片中如何分布（即使数据完全随机分布）都可正常工作。添加新分片时，不必将旧数据传输到该分片。你可以给新分片分配大权重然后写新数据 - 数据可能会稍分布不均，但查询会正确高效地运行。

下面的情况，你需要关注分片方案：

- 使用需要特定键连接数据（**IN** 或 **JOIN**）的查询。如果数据是用该键进行分片，则应使用本地 **IN** 或 **JOIN** 而不是 **GLOBAL IN** 或 **GLOBAL JOIN**，这样效率更高。
- 使用大量服务器（上百或更多），但有大量小查询（个别客户的查询 - 网站，广告商或合作伙伴）。为了使小查询不影响整个集群，让单个客户的数据处于单个分片上是有意义的。或者，正如我们在 Yandex.Metrica 中所做的那样，你可以配置两级分片：将整个集群划分为«层»，一个层可以包含多个分片。单个客户的数据位于单个层上，根据需要将分片添加到层中，层中的数据随机分布。然后给每层创建分布式表，再创建一个全局的分布式表用于全局的查询。

数据是异步写入的。对于分布式表的 **INSERT**，数据块只写本地文件系统。之后会尽快地在后台发送到远程服务器。你可以通过查看表目录中的文件列表（等待发送的数据）来检查数据是否成功发送：`/var/lib/clickhouse/data/database/table/`。

如果在 **INSERT** 到分布式表时服务器节点丢失或重启（如，设备故障），则插入的数据可能会丢失。如果在表目录中检测到损坏的数据分片，则会将其转移到«broken»子目录，并不再使用。

启用 **max_parallel_replicas** 选项后，会在分表的所有副本上并行查询处理。更多信息，请参阅«设置，**max_parallel_replicas**»部分。

加入我们

加载好的 **JOIN** 表数据会常驻内存中。

Join(ANY|ALL, LEFT|INNER, k1[, k2, ...])

引擎参数：**ANY|ALL** – 连接修饰；**LEFT|INNER** – 连接类型。更多信息可参考 [JOIN子句](#)。

这些参数设置不用带引号，但必须与要 **JOIN** 表匹配。**k1, k2,** 是 **USING** 子句中要用于连接的关键列。

此引擎表不能用于 **GLOBAL JOIN**。

类似于 **Set** 引擎，可以使用 **INSERT** 向表中添加数据。设置为 **ANY** 时，重复键的数据会被忽略（仅一条用于连接）。设置为 **ALL** 时，重复键的数据都会用于连接。不能直接对 **JOIN** 表进行 **SELECT**。检索其数据的唯一方法是将其作为 **JOIN** 语句右边的表。

跟 **Set** 引擎类似，**Join** 引擎把数据存储在磁盘中。

限制和设置

创建表时，将应用以下设置：

- **join_use_nulls**
- **max_rows_in_join**
- **max_bytes_in_join**
- **join_overflow_mode**
- **join_any_take_last_row**

该 **Join**-发动机表不能用于 **GLOBAL JOIN** 操作。

合并

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

Merge 引擎的参数：一个数据库名和一个用于匹配表名的正则表达式。

示例：

```
Merge(hits, '^WatchLog')
```

数据会从 `hits` 数据库中表名匹配正则 '`^WatchLog`' 的表中读取。

除了数据库名，你也可以用一个返回字符串的常量表达式。例如，`currentDatabase()`。

正则表达式 — `re2` (支持 PCRE 一个子集的功能)，大小写敏感。

了解关于正则表达式中转义字符的说明可参看 «match» 一节。

当选择需要读的表时，`Merge` 表本身会被排除，即使它匹配上了该正则。这样设计为了避免循环。

当然，是能够创建两个相互无限递归读取对方数据的 `Merge` 表的，但这并没有什么意义。

`Merge` 引擎的一个典型应用是可以像使用一张表一样使用大量的 `TinyLog` 表。

示例 2：

我们假定你有一个旧表 (`WatchLog_old`)，你想改变数据分区了，但又不想把旧数据转移到新表 (`WatchLog_new`) 里，并且你需要同时能看到这两个表的数据。

```
CREATE TABLE WatchLog_old(date Date, UserId Int64, EventType String, Cnt UInt64)
ENGINE=MergeTree(date, (UserId, EventType), 8192);
INSERT INTO WatchLog_old VALUES ('2018-01-01', 1, 'hit', 3);

CREATE TABLE WatchLog_new(date Date, UserId Int64, EventType String, Cnt UInt64)
ENGINE=MergeTree PARTITION BY date ORDER BY (UserId, EventType) SETTINGS index_granularity=8192;
INSERT INTO WatchLog_new VALUES ('2018-01-02', 2, 'hit', 3);

CREATE TABLE WatchLog as WatchLog_old ENGINE=Merge(currentDatabase(), '^WatchLog');

SELECT *
FROM WatchLog

date UserId EventType Cnt
2018-01-01 | 1 | hit | 3 |
              |   |   |   |
date UserId EventType Cnt
2018-01-02 | 2 | hit | 3 |
              |   |   |   |
```

虚拟列

虚拟列是一种由表引擎提供而不是在表定义中的列。换种说法就是，这些列并没有在 `CREATE TABLE` 中指定，但可以在 `SELECT` 中使用。

下面列出虚拟列跟普通列的不同点：

- 虚拟列不在表结构定义里指定。
- 不能用 `INSERT` 向虚拟列写数据。
- 使用不指定列名的 `INSERT` 语句时，虚拟列要会被忽略掉。
- 使用星号通配符 (`SELECT *`) 时虚拟列不会包含在里面。
- 虚拟列不会出现在 `SHOW CREATE TABLE` 和 `DESC TABLE` 的查询结果里。

`Merge` 类型的表包括一个 `String` 类型的 `_table` 虚拟列。（如果该表本来已有了一个 `_table` 的列，那这个虚拟列会命名为 `_table1`；如果 `_table1` 也本就存在了，那这个虚拟列会被命名为 `_table2`，依此类推）该列包含被读数据的表名。

如果 `WHERE/PREWHERE` 子句包含了带 `_table` 的条件，并且没有依赖其他的列（如作为表达式谓词链接的一个子项或作为整个的表达式），这些条件的作用会像索引一样。这些条件会在那些可能被读数据的表的表名上执行，并且读操作只会在那些满

足了该条件的表上去执行。

字典

Dictionary 引擎将字典数据展示为一个ClickHouse的表。

例如，考虑使用一个具有以下配置的 products 字典：

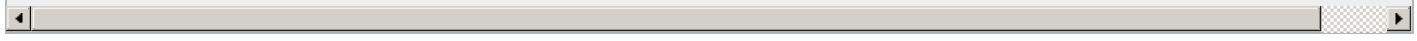
```
<dictionaries>
<dictionary>
  <name>products</name>
  <source>
    <odbc>
      <table>products</table>
      <connection_string>DSN=some-db-server</connection_string>
    </odbc>
  </source>
  <lifetime>
    <min>300</min>
    <max>360</max>
  </lifetime>
  <layout>
    <flat/>
  </layout>
  <structure>
    <id>
      <name>product_id</name>
    </id>
    <attribute>
      <name>title</name>
      <type>String</type>
      <null_value></null_value>
    </attribute>
  </structure>
</dictionary>
</dictionaries>
```

查询字典中的数据：

```
select name, type, key, attribute.names, attribute.types, bytes_allocated, element_count, source from system.dictionaries where name = 'products';
```

```
SELECT
  name,
  type,
  key,
  attribute.names,
  attribute.types,
  bytes_allocated,
  element_count,
  source
FROM system.dictionaries
WHERE name = 'products'
```

name	type	key	attribute.names	attribute.types	bytes_allocated	element_count	source
products	Flat	UInt64	['title']	['String']	23065376	175032	ODBC: .products



你可以使用 `dictGet*` 函数来获取这种格式的字典数据。

当你需要获取原始数据，或者是想要使用 `JOIN` 操作的时候，这种视图并没有什么帮助。对于这些情况，你可以使用 `Dictionary` 引擎，它可以将字典数据展示在表中。

语法：

```
CREATE TABLE %table_name% (%fields%) engine = Dictionary(%dictionary_name%)`
```

示例：

```
create table products (product_id UInt64, title String) Engine = Dictionary(products);

CREATE TABLE products
(
    product_id UInt64,
    title String,
)
ENGINE = Dictionary(products)
```

Ok.

0 rows in set. Elapsed: 0.004 sec.

看一看表中的内容。

```
select * from products limit 1;

SELECT *
FROM products
LIMIT 1
```

product_id	title
152689	Some item

1 rows in set. Elapsed: 0.006 sec.

文件(输入格式)

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。

用法示例：

- 从 ClickHouse 导出数据到文件。
- 将数据从一种格式转换为另一种格式。
- 通过编辑磁盘上的文件来更新 ClickHouse 中的数据。

在 ClickHouse 服务器中的使用

File(Format)

选用的 `Format` 需要支持 `INSERT` 或 `SELECT`。有关支持格式的完整列表，请参阅 [格式](#)。

ClickHouse 不支持给 `File` 指定文件系统路径。它使用服务器配置中 `路径` 设定的文件夹。

使用 `File(Format)` 创建表时，它会在该文件夹中创建空的子目录。当数据写入该表时，它会写到该子目录中的 `data.Format` 文件中。

你也可以在服务器文件系统中手动创建这些子文件夹和文件，然后通过 `ATTACH` 将其创建为具有对应名称的表，这样你就可以从该文件中查询数据了。

注意

注意这个功能，因为 ClickHouse 不会跟踪这些文件在外部的更改。在 ClickHouse 中和 ClickHouse 外部同时写入会造成结果是不确定的。

示例：

1. 创建 `file_engine_table` 表：

```
CREATE TABLE file_engine_table (name String, value UInt32) ENGINE=File(TabSeparated)
```

默认情况下，Clickhouse 会创建目录 `/var/lib/clickhouse/data/default/file_engine_table`。

2. 手动创建 `/var/lib/clickhouse/data/default/file_engine_table/data.TabSeparated` 文件，并且包含内容：

```
$ cat data.TabSeparated
one 1
two 2
```

3. 查询这些数据：

```
SELECT * FROM file_engine_table
```

name	value
one	1
two	2

在 Clickhouse-local 中的使用

使用 `环板-ヨツ嘉ツツ偲` 时，File 引擎除了 `Format` 之外，还可以接受文件路径参数。可以使用数字或人类可读的名称来指定标准输入/输出流，例如 `0` 或 `stdin`，`1` 或 `stdout`。

例如：

```
$ echo -e "1,2\n3,4" | clickhouse-local -q "CREATE TABLE table (a Int64, b Int64) ENGINE = File(CSV, stdin); SELECT a, b FROM table; DROP TABLE table"
```

功能实现

- 读操作可支持并发，但写操作不支持
- 不支持：

- `ALTER`
- `SELECT ... SAMPLE`
- `委引`

- 小节
- 副本

查看

用于构建视图（有关更多信息，请参阅 `CREATE VIEW` 查询）。它不存储数据，仅存储指定的 `SELECT` 查询。从表中读取时，它会运行此查询（并从查询中删除所有不必要的列）。

用于查询处理的外部数据

ClickHouse 允许向服务器发送处理查询所需的数据以及 `SELECT` 查询。这些数据放在一个临时表中（请参阅《临时表》一节），可以在查询中使用（例如，在 `IN` 操作符中）。

例如，如果您有一个包含重要用户标识符的文本文件，则可以将其与使用此列表过滤的查询一起上传到服务器。

如果需要使用大量外部数据运行多个查询，请不要使用该特性。最好提前把数据上传到数据库。

可以使用命令行客户端（在非交互模式下）或使用 HTTP 接口上传外部数据。

在命令行客户端中，您可以指定格式的参数部分

```
--external --file=... [--name=...] [--format=...] [--types=...|--structure=...]
```

对于传输的表的数量，可能有多个这样的部分。

-external – 标记子句的开始。

-file – 带有表存储的文件的路径，或者，它指的是STDIN。

只能从 `stdin` 中检索单个表。

以下的参数是可选的：**-name** – 表的名称，如果省略，则采用 `_data`。

-format – 文件中的数据格式。如果省略，则使用 `TabSeparated`。

以下的参数必选一个：**-types** – 逗号分隔列类型的列表。例如：`UInt64,String`。列将被命名为 `_1`, `_2`, ...

-structure – 表结构的格式 `UserID UInt64 , URL String`。定义列的名字以及类型。

在《file》中指定的文件将由《format》中指定的格式解析，使用在《types》或《structure》中指定的数据类型。该表将被上传到服务器，并在作为名称为《name》临时表。

示例：

```
echo -ne "1\n2\n3\n" | clickhouse-client --query="SELECT count() FROM test.visits WHERE TraficSourceID IN _data" --external --file=- --types=Int8  
849897  
cat /etc/passwd | sed 's/:/\t/g' | clickhouse-client --query="SELECT shell, count() AS c FROM passwd GROUP BY shell ORDER BY c DESC" --external --file=- --name=passwd --structure='login String, unused String, uid UInt16, gid UInt16, comment String, home String, shell String'  
/bin/sh 20  
/bin/false 5  
/bin/bash 4  
/usr/sbin/nologin 1  
/bin/sync 1
```

当使用HTTP接口时，外部数据以 `multipart/form-data` 格式传递。每个表作为一个单独的文件传输。表名取自文件名。《query_string》传递参数《name_format》、《name_types》和《name_structure》，其中《name》是这些参数对应的表的名称。参数的含义与使用命令行客户端时的含义相同。

示例：

```
cat /etc/passwd | sed 's/:/\t/g' > passwd.tsv
```

```
curl -F 'passwd=@passwd.tsv;' 'http://localhost:8123/?query=SELECT+shell,+count() AS c+FROM+passwd+GROUP+BY+shell+ORDER+BY+c+DESC&passwd_structure=log in+String,+unused+String,+uid+UInt16,+gid+UInt16,+comment+String,+home+String,+shell+String' /bin/sh 20 /bin/false    5 /bin/bash      4 /usr/sbin/nologin  1 /bin/sync      1
```

对于分布式查询，将临时表发送到所有远程服务器。

缓冲区

缓冲数据写入 RAM 中，周期性地将数据刷新到另一个表。在读取操作时，同时从缓冲区和另一个表读取数据。

```
Buffer(database, table, num_layers, min_time, max_time, min_rows, max_rows, min_bytes, max_bytes)
```

引擎的参数 : `database`, `table` - 要刷新数据的表。可以使用返回字符串的常量表达式而不是数据库名称。`num_layers` - 并行层数。在物理上，该表将表示为 `num_layers` 个独立缓冲区。建议值为 16。

`min_time`, `max_time`, `min_rows`, `max_rows`, `min_bytes`, `max_bytes` - 从缓冲区刷新数据的条件。

如果满足所有 «`min`» 条件或至少一个 «`max`» 条件，则从缓冲区刷新数据并将其写入目标表。`min_time`, `max_time` — 从第一次写入缓冲区时起以秒为单位的时间条件。`min_rows`, `max_rows` - 缓冲区中行数的条件。`min_bytes`, `max_bytes` - 缓冲区中字节数的条件。

写入时，数据从 `num_layers` 个缓冲区中随机插入。或者，如果插入数据的大小足够大（大于 `max_rows` 或 `max_bytes`），则会绕过缓冲区将其写入目标表。

每个 «`num_layers`» 缓冲区刷新数据的条件是分别计算。例如，如果 `num_layers = 16` 且 `max_bytes = 1000000000`，则最大RAM消耗将为 1.6 GB。

示例：

```
CREATE TABLE merge.hits_buffer AS merge.hits ENGINE = Buffer(merge, hits, 16, 10, 100, 10000, 1000000, 10000000, 100000000)
```

创建一个 «`merge.hits_buffer`» 表，其结构与 «`merge.hits`» 相同，并使用 Buffer 引擎。写入此表时，数据缓冲在 RAM 中，然后写入 «`merge.hits`» 表。创建了 16 个缓冲区。如果已经过了 100 秒，或者已写入 100 万行，或者已写入 100 MB 数据，则刷新每个缓冲区的数据；或者如果同时已经过了 10 秒并且已经写入了 10,000 行和 10 MB 的数据。例如，如果只写了一行，那么在 100 秒之后，都会被刷新。但是如果写了很多行，数据将会更快地刷新。

当服务器停止时，使用 `DROP TABLE` 或 `DETACH TABLE`，缓冲区数据也会刷新到目标表。

可以为数据库和表名在单个引号中设置空字符串。这表示没有目的地表。在这种情况下，当达到数据刷新条件时，缓冲器被简单地清除。这可能对于保持数据窗口在内存中是有用的。

从 Buffer 表读取时，将从缓冲区和目标表（如果有）处理数据。

请注意，Buffer 表不支持索引。换句话说，缓冲区中的数据被完全扫描，对于大缓冲区来说可能很慢。（对于目标表中的数据，将使用它支持的索引。）

如果 Buffer 表中的列集与目标表中的列集不匹配，则会插入两个表中存在的列的子集。

如果类型与 Buffer 表和目标表中的某列不匹配，则会在服务器日志中输入错误消息并清除缓冲区。

如果在刷新缓冲区时目标表不存在，则会发生同样的情况。

如果需要为目标表和 Buffer 表运行 `ALTER`，我们建议先删除 Buffer 表，为目标表运行 `ALTER`，然后再次创建 Buffer 表。

如果服务器异常重启，缓冲区中的数据将丢失。

PREWHERE, **FINAL** 和 **SAMPLE** 对缓冲表不起作用。这些条件将传递到目标表，但不用于处理缓冲区中的数据。因此，我们建议只使用 Buffer 表进行写入，同时从目标表进行读取。

将数据添加到缓冲区时，其中一个缓冲区被锁定。如果同时从表执行读操作，则会导致延迟。

插入到 Buffer 表中的数据可能以不同的顺序和不同的块写入目标表中。因此，Buffer 表很难用于正确写入 **CollapsingMergeTree**。为避免出现问题，您可以将 `«num_layers»` 设置为 1。

如果目标表是复制表，则在写入 Buffer 表时会丢失复制表的某些预期特征。数据部分的行次序和大小的随机变化导致数据不能去重，这意味着无法对复制表进行可靠的 `«exactly once»` 写入。

由于这些缺点，我们只建议在极少数情况下使用 Buffer 表。

当在单位时间内从大量服务器接收到太多 **INSERTs** 并且在插入之前无法缓冲数据时使用 Buffer 表，这意味着这些 **INSERTs** 不能足够快地执行。

请注意，一次插入一行数据是没有意义的，即使对于 Buffer 表也是如此。这将只产生每秒几千行的速度，而插入更大的数据块每秒可以产生超过一百万行（参见 `«性能»` 部分）。

记忆

Memory 引擎以未压缩的形式将数据存储在 RAM 中。数据完全以读取时获得的形式存储。换句话说，从这张表中读取是很轻松的。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。阅读是并行化的。在简单查询上达到最大生产率（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）。重新启动服务器时，表中的数据消失，表将变为空。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。

Memory 引擎是由系统用于临时表进行外部数据的查询（请参阅 `«外部数据用于请求处理»` 部分），以及用于实现 **GLOBAL IN**（请参见 `«IN 运算符»` 部分）。

设置

始终存在于 RAM 中的数据集。它适用于 **IN** 运算符的右侧（请参见 `«IN 运算符»` 部分）。

可以使用 **INSERT** 向表中插入数据。新元素将添加到数据集中，而重复项将被忽略。但是不能对此类型表执行 **SELECT** 语句。检索数据的唯一方法是在 **IN** 运算符的右半部分使用它。

数据始终存在于 RAM 中。对于 **INSERT**，插入数据块也会写入磁盘上的表目录。启动服务器时，此数据将加载到 RAM。也就是说，重新启动后，数据仍然存在。

对于强制服务器重启，磁盘上的数据块可能会丢失或损坏。在数据块损坏的情况下，可能需要手动删除包含损坏数据的文件。

表引擎

表引擎（即表的类型）决定了：

- 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。

引擎类型

MergeTree

适用于高负载任务的最通用和功能最强大的表引擎。这些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。

该类型的引擎：

- [MergeTree](#)
- [ReplacingMergeTree](#)
- [SummingMergeTree](#)
- [AggregatingMergeTree](#)
- [CollapsingMergeTree](#)
- [VersionedCollapsingMergeTree](#)
- [GraphiteMergeTree](#)

日志

具有最小功能的[轻量级引擎](#)。当您需要快速写入许多小表（最多约100万行）并在以后整体读取它们时，该类型的引擎是最有效的。

该类型的引擎：

- [TinyLog](#)
- [StripeLog](#)
- [Log](#)

集成引擎

用于与其他的数据存储与处理系统集成的引擎。

该类型的引擎：

- [Kafka](#)
- [MySQL](#)
- [ODBC](#)
- [JDBC](#)
- [HDFS](#)

用于其他特定功能的引擎

该类型的引擎：

- [Distributed](#)
- [MaterializedView](#)
- [Dictionary](#)
- [\[Merge\]\(special/merge.md#merge\)](#)
- [File](#)
- [Null](#)
- [Set](#)
- [Join](#)
- [URL](#)
- [View](#)
- [Memory](#)
- [Buffer](#)

虚拟列

虚拟列是表引擎组成的一部分，它在对应的表引擎的源代码中定义。

您不能在 `CREATE TABLE` 中指定虚拟列，并且虚拟列不会包含在 `SHOW CREATE TABLE` 和 `DESCRIBE TABLE` 的查询结果中。虚拟列是只读的，所以您不能向虚拟列中写入数据。

如果想要查询虚拟列中的数据，您必须在 `SELECT` 查询中包含虚拟列的名字。`SELECT *` 不会返回虚拟列的内容。

若您创建的表中有一列与虚拟列的名字相同，那么虚拟列将不能再被访问。我们不建议您这样做。为了避免这种列名的冲突，虚拟列的名字一般都以下划线开头。

SQL参考

ClickHouse 支持以下类型的查询：

- [SELECT](#)
- [INSERT INTO](#)
- [CREATE](#)
- [ALTER](#)
- [其他类型的查询](#)

SELECT Queries Syntax

`SELECT` performs data retrieval.

```
[WITH expr_list|(subquery)]
SELECT [DISTINCT] expr_list
[FROM [db.]table | (subquery) | table_function] [FINAL]
[SAMPLE sample_coeff]
[ARRAY JOIN ...]
[GLOBAL] [ANY|ALL] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER] JOIN (subquery)|table USING columns_list
[PREWHERE expr]
[WHERE expr]
[GROUP BY expr_list] [WITH TOTALS]
[HAVING expr]
[ORDER BY expr_list]
[LIMIT [offset_value, ]n BY columns]
[LIMIT [n, ]m]
[UNION ALL ...]
[INTO OUTFILE filename]
[FORMAT format]
```

All clauses are optional, except for the required list of expressions immediately after `SELECT` which is covered in more detail [below](#).

Specifics of each optional clause are covered in separate sections, which are listed in the same order as they are executed:

- [WITH clause](#)
- [FROM clause](#)
- [SAMPLE clause](#)
- [JOIN clause](#)
- [PREWHERE clause](#)
- [WHERE clause](#)
- [GROUP BY clause](#)
- [LIMIT BY clause](#)
- [HAVING clause](#)
- [SELECT clause](#)
- [DISTINCT clause](#)
- [LIMIT clause](#)
- [UNION ALL clause](#)

- **INTO OUTFILE clause**
- **FORMAT clause**

SELECT Clause

Expressions specified in the `SELECT` clause are calculated after all the operations in the clauses described above are finished. These expressions work as if they apply to separate rows in the result. If expressions in the `SELECT` clause contain aggregate functions, then ClickHouse processes aggregate functions and expressions used as their arguments during the **GROUP BY** aggregation.

If you want to include all columns in the result, use the asterisk (*) symbol. For example, `SELECT * FROM ...`

To match some columns in the result with a **re2** regular expression, you can use the `COLUMNS` expression.

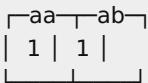
```
COLUMNS('regexp')
```

For example, consider the table:

```
CREATE TABLE default.col_names (aa Int8, ab Int8, bc Int8) ENGINE = TinyLog
```

The following query selects data from all the columns containing the `a` symbol in their name.

```
SELECT COLUMNS('a') FROM col_names
```

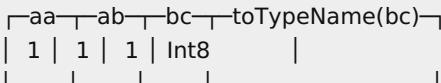


The selected columns are returned not in the alphabetical order.

You can use multiple `COLUMNS` expressions in a query and apply functions to them.

For example:

```
SELECT COLUMNS('a'), COLUMNS('c'), toTypeName(COLUMNS('c')) FROM col_names
```



Each column returned by the `COLUMNS` expression is passed to the function as a separate argument. Also you can pass other arguments to the function if it supports them. Be careful when using functions. If a function doesn't support the number of arguments you have passed to it, ClickHouse throws an exception.

For example:

```
SELECT COLUMNS('a') + COLUMNS('c') FROM col_names
```

Received exception from server (version 19.14.1):

Code: 42. DB::Exception: Received from localhost:9000. DB::Exception: Number of arguments for function plus doesn't match: passed 3, should be 2.

In this example, `COLUMNS('a')` returns two columns: `aa` and `ab`. `COLUMNS('c')` returns the `bc` column. The `+` operator can't apply to 3 arguments, so ClickHouse throws an exception with the relevant message.

Columns that matched the `COLUMNS` expression can have different data types. If `COLUMNS` doesn't match any columns and is the only expression in `SELECT`, ClickHouse throws an exception.

Asterisk

You can put an asterisk in any part of a query instead of an expression. When the query is analyzed, the asterisk is expanded to a list of all table columns (excluding the `MATERIALIZED` and `ALIAS` columns). There are only a few cases when using an asterisk is justified:

- When creating a table dump.
- For tables containing just a few columns, such as system tables.
- For getting information about what columns are in a table. In this case, set `LIMIT 1`. But it is better to use the `DESC TABLE` query.
- When there is strong filtration on a small number of columns using `PREWHERE`.
- In subqueries (since columns that aren't needed for the external query are excluded from subqueries).

In all other cases, we don't recommend using the asterisk, since it only gives you the drawbacks of a columnar DBMS instead of the advantages. In other words using the asterisk is not recommended.

Extreme Values

In addition to results, you can also get minimum and maximum values for the results columns. To do this, set the `extremes` setting to 1. Minimums and maximums are calculated for numeric types, dates, and dates with times. For other columns, the default values are output.

An extra two rows are calculated – the minimums and maximums, respectively. These extra two rows are output in `JSON*`, `TabSeparated*`, and `Pretty*` `formats`, separate from the other rows. They are not output for other formats.

In `JSON*` formats, the extreme values are output in a separate 'extremes' field. In `TabSeparated*` formats, the row comes after the main result, and after 'totals' if present. It is preceded by an empty row (after the other data). In `Pretty*` formats, the row is output as a separate table after the main result, and after `totals` if present.

Extreme values are calculated for rows before `LIMIT`, but after `LIMIT BY`. However, when using `LIMIT offset, size`, the rows before `offset` are included in `extremes`. In stream requests, the result may also include a small number of rows that passed through `LIMIT`.

Notes

You can use synonyms (`AS` aliases) in any part of a query.

The `GROUP BY` and `ORDER BY` clauses do not support positional arguments. This contradicts MySQL, but conforms to standard SQL. For example, `GROUP BY 1, 2` will be interpreted as grouping by constants (i.e. aggregation of all rows into one).

Implementation Details

If the query omits the `DISTINCT`, `GROUP BY` and `ORDER BY` clauses and the `IN` and `JOIN` subqueries, the query will be completely stream processed, using O(1) amount of RAM. Otherwise, the query might consume a lot of RAM if the appropriate restrictions are not specified:

- `max_memory_usage`
- `max_rows_to_group_by`
- `max_rows_to_sort`
- `max_rows_in_distinct`
- `max_bytes_in_distinct`

- max_rows_in_set
- max_bytes_in_set
- max_rows_in_join
- max_bytes_in_join
- max_bytes_before_external_sort
- max_bytes_before_external_group_by

For more information, see the section “Settings”. It is possible to use external sorting (saving temporary tables to a disk) and external aggregation.

ARRAY JOIN Clause

It is a common operation for tables that contain an array column to produce a new table that has a column with each individual array element of that initial column, while values of other columns are duplicated. This is the basic case of what `ARRAY JOIN` clause does.

Its name comes from the fact that it can be looked at as executing `JOIN` with an array or nested data structure. The intent is similar to the `arrayJoin` function, but the clause functionality is broader.

Syntax:

```
SELECT <expr_list>
FROM <left_subquery>
[LEFT] ARRAY JOIN <array>
[WHERE|PREWHERE <expr>]
...
...
```

You can specify only one `ARRAY JOIN` clause in a `SELECT` query.

Supported types of `ARRAY JOIN` are listed below:

- `ARRAY JOIN` - In base case, empty arrays are not included in the result of `JOIN`.
- `LEFT ARRAY JOIN` - The result of `JOIN` contains rows with empty arrays. The value for an empty array is set to the default value for the array element type (usually 0, empty string or NULL).

Basic ARRAY JOIN Examples

The examples below demonstrate the usage of the `ARRAY JOIN` and `LEFT ARRAY JOIN` clauses. Let's create a table with an `Array` type column and insert values into it:

```
CREATE TABLE arrays_test
(
    s String,
    arr Array(UInt8)
) ENGINE = Memory;

INSERT INTO arrays_test
VALUES ('Hello', [1,2]), ('World', [3,4,5]), ('Goodbye', []);
```

s	arr
Hello	[1,2]
World	[3,4,5]
Goodbye	[]

The example below uses the `ARRAY JOIN` clause:

```
SELECT s, arr
FROM arrays_test
ARRAY JOIN arr;
```

s	arr
Hello	1
Hello	2
World	3
World	4
World	5

The next example uses the `LEFT ARRAY JOIN` clause:

```
SELECT s, arr
FROM arrays_test
LEFT ARRAY JOIN arr;
```

s	arr
Hello	1
Hello	2
World	3
World	4
World	5
Goodbye	0

Using Aliases

An alias can be specified for an array in the `ARRAY JOIN` clause. In this case, an array item can be accessed by this alias, but the array itself is accessed by the original name. Example:

```
SELECT s, arr, a
FROM arrays_test
ARRAY JOIN arr AS a;
```

s	arr	a
Hello	[1,2]	1
Hello	[1,2]	2
World	[3,4,5]	3
World	[3,4,5]	4
World	[3,4,5]	5

Using aliases, you can perform `ARRAY JOIN` with an external array. For example:

```
SELECT s, arr_external
FROM arrays_test
ARRAY JOIN [1, 2, 3] AS arr_external;
```

s	arr_external
Hello	1
Hello	2
Hello	3
World	1
World	2
World	3
Goodbye	1
Goodbye	2
Goodbye	3

Multiple arrays can be comma-separated in the `ARRAY JOIN` clause. In this case, `JOIN` is performed with them simultaneously (the direct sum, not the cartesian product). Note that all the arrays must have the same size. Example:

```
SELECT s, arr, a, num, mapped
FROM arrays_test
ARRAY JOIN arr AS a, arrayEnumerate(arr) AS num, arrayMap(x -> x + 1, arr) AS mapped;
```

s	arr	a	num	mapped
Hello	[1,2]	1	1	2
Hello	[1,2]	2	2	3
World	[3,4,5]	3	1	4
World	[3,4,5]	4	2	5
World	[3,4,5]	5	3	6

The example below uses the `arrayEnumerate` function:

```
SELECT s, arr, a, num, arrayEnumerate(arr)
FROM arrays_test
ARRAY JOIN arr AS a, arrayEnumerate(arr) AS num;
```

s	arr	a	num	arrayEnumerate(arr)
Hello	[1,2]	1	1	[1,2]
Hello	[1,2]	2	2	[1,2]
World	[3,4,5]	3	1	[1,2,3]
World	[3,4,5]	4	2	[1,2,3]
World	[3,4,5]	5	3	[1,2,3]

ARRAY JOIN with Nested Data Structure

`ARRAY JOIN` also works with [nested data structures](#):

```
CREATE TABLE nested_test
(
    s String,
    nest Nested(
        x UInt8,
        y UInt32)
) ENGINE = Memory;

INSERT INTO nested_test
VALUES ('Hello', [1,2], [10,20], [1,2,3]), ('World', [3,4,5], [10,20,30], [1,2,3,4,5]), ('Goodbye', [5,6], [10,20,30,40], [1,2,3,4,5,6]);
```

```
VALUES ('Hello', [1,2], [10,20]), ('World', [3,4,5], [30,40,50]), ('Goodbye', [], []);
```

s			nest.x	nest.y
Hello	[1,2]		[10,20]	
World		[3,4,5]		[30,40,50]
Goodbye	[]		[]	

```
SELECT s, `nest.x`, `nest.y`  
FROM nested_test  
ARRAY JOIN nest;
```

s			nest.x	nest.y
Hello	1		10	
Hello	2		20	
World	3		30	
World	4		40	
World	5		50	

When specifying names of nested data structures in `ARRAY JOIN`, the meaning is the same as `ARRAY JOIN` with all the array elements that it consists of. Examples are listed below:

```
SELECT s, `nest.x`, `nest.y`  
FROM nested_test  
ARRAY JOIN `nest.x`, `nest.y`;
```

s			nest.x	nest.y
Hello	1		10	
Hello	2		20	
World	3		30	
World	4		40	
World	5		50	

This variation also makes sense:

```
SELECT s, `nest.x`, `nest.y`  
FROM nested_test  
ARRAY JOIN `nest.x`;
```

s			nest.x	nest.y
Hello	1		[10,20]	
Hello	2		[10,20]	
World	3		[30,40,50]	
World	4		[30,40,50]	
World	5		[30,40,50]	

An alias may be used for a nested data structure, in order to select either the `JOIN` result or the source array. Example:

```
SELECT s, `n.x`, `n.y`, `nest.x`, `nest.y`  
FROM nested_test  
ARRAY JOIN nest AS n;
```

s	n.x	n.y	nest.x	nest.y
Hello	1	10	[1,2]	[10,20]
Hello	2	20	[1,2]	[10,20]
World	3	30	[3,4,5]	[30,40,50]
World	4	40	[3,4,5]	[30,40,50]
World	5	50	[3,4,5]	[30,40,50]

Example of using the `arrayEnumerate` function:

```
SELECT s, `n.x`, `n.y`, `nest.x`, `nest.y`, num  
FROM nested_test  
ARRAY JOIN nest AS n, arrayEnumerate(`nest.x`) AS num;
```

s	n.x	n.y	nest.x	nest.y	num
Hello	1	10	[1,2]	[10,20]	1
Hello	2	20	[1,2]	[10,20]	2
World	3	30	[3,4,5]	[30,40,50]	1
World	4	40	[3,4,5]	[30,40,50]	2
World	5	50	[3,4,5]	[30,40,50]	3

Implementation Details

The query execution order is optimized when running `ARRAY JOIN`. Although `ARRAY JOIN` must always be specified before the `WHERE/PREWHERE` clause in a query, technically they can be performed in any order, unless result of `ARRAY JOIN` is used for filtering. The processing order is controlled by the query optimizer.

DISTINCT Clause

If `SELECT DISTINCT` is specified, only unique rows will remain in a query result. Thus only a single row will remain out of all the sets of fully matching rows in the result.

Null Processing

`DISTINCT` works with `NULL` as if `NULL` were a specific value, and `NULL==NULL`. In other words, in the `DISTINCT` results, different combinations with `NULL` occur only once. It differs from `NULL` processing in most other contexts.

Alternatives

It is possible to obtain the same result by applying `GROUP BY` across the same set of values as specified as `SELECT` clause, without using any aggregate functions. But there are few differences from `GROUP BY` approach:

- `DISTINCT` can be applied together with `GROUP BY`.
- When `ORDER BY` is omitted and `LIMIT` is defined, the query stops running immediately after the required number of different rows has been read.
- Data blocks are output as they are processed, without waiting for the entire query to finish running.

Limitations

`DISTINCT` is not supported if `SELECT` has at least one array column.

Examples

ClickHouse supports using the `DISTINCT` and `ORDER BY` clauses for different columns in one query. The `DISTINCT` clause is executed before the `ORDER BY` clause.

Example table:

a	b
2	1
1	2
3	3
2	4

When selecting data with the `SELECT DISTINCT a FROM t1 ORDER BY b ASC` query, we get the following result:

a
2
1
3

If we change the sorting direction `SELECT DISTINCT a FROM t1 ORDER BY b DESC`, we get the following result:

a
3
1
2

Row 2, 4 was cut before sorting.

Take this implementation specificity into account when programming queries.

FORMAT Clause

ClickHouse supports a wide range of [serialization formats](#) that can be used on query results among other things. There are multiple ways to choose a format for `SELECT` output, one of them is to specify `FORMAT` at the end of query to get resulting data in any specific format.

Specific format might be used either for convenience, integration with other systems or performance gain.

Default Format

If the `FORMAT` clause is omitted, the default format is used, which depends on both the settings and the interface used for accessing the ClickHouse server. For the [HTTP interface](#) and the [command-line client](#) in batch mode, the default format is `TabSeparated`. For the command-line client in interactive mode, the default format is `PrettyCompact` (it produces compact human-readable tables).

Implementation Details

When using the command-line client, data is always passed over the network in an internal efficient format (`Native`). The client independently interprets the `FORMAT` clause of the query and formats the data itself (thus relieving the network and the server from the extra load).

FROM Clause

The `FROM` clause specifies the source to read data from:

- [Table](#)
- [Subquery](#)
- [Table function](#)

[JOIN](#) and [ARRAY JOIN](#) clauses may also be used to extend the functionality of the `FROM` clause.

Subquery is another `SELECT` query that may be specified in parenthesis inside `FROM` clause.

`FROM` clause can contain multiple data sources, separated by commas, which is equivalent of performing [CROSS JOIN](#) on them.

FINAL Modifier

When `FINAL` is specified, ClickHouse fully merges the data before returning the result and thus performs all data transformations that happen during merges for the given table engine.

It is applicable when selecting data from tables that use the [MergeTree-engine family](#) (except [GraphiteMergeTree](#)). Also supported for:

- [Replicated](#) versions of MergeTree engines.
- [View](#), [Buffer](#), [Distributed](#), and [MaterializedView](#) engines that operate over other engines, provided they were created over `MergeTree`-engine tables.

Drawbacks

Queries that use `FINAL` are executed not as fast as similar queries that don't, because:

- Query is executed in a single thread and data is merged during query execution.
- Queries with `FINAL` read primary key columns in addition to the columns specified in the query.

In most cases, avoid using `FINAL`. The common approach is to use different queries that assume the background processes of the `MergeTree` engine haven't happened yet and deal with it by applying aggregation (for example, to discard duplicates).

Implementation Details

If the `FROM` clause is omitted, data will be read from the `system.one` table.

The `system.one` table contains exactly one row (this table fulfills the same purpose as the DUAL table found in other DBMSs).

To execute a query, all the columns listed in the query are extracted from the appropriate table. Any columns not needed for the external query are thrown out of the subqueries.

If a query does not list any columns (for example, `SELECT count() FROM t`), some column is extracted from the table anyway (the smallest one is preferred), in order to calculate the number of rows.

GROUP BY Clause

`GROUP BY` clause switches the `SELECT` query into an aggregation mode, which works as follows:

- `GROUP BY` clause contains a list of expressions (or a single expression, which is considered to be the list of length one). This list acts as a "grouping key", while each individual expression will be referred to as a "key expressions".
- All the expressions in the [SELECT](#), [HAVING](#), and [ORDER BY](#) clauses **must** be calculated based on key expressions **or** on [aggregate functions](#) over non-key expressions (including plain columns). In other words, each column selected from the table must be used either in a key expression or inside an

aggregate function, but not both.

- Result of aggregating `SELECT` query will contain as many rows as there were unique values of “grouping key” in source table. Usually this significantly reduces the row count, often by orders of magnitude, but not necessarily: row count stays the same if all “grouping key” values were distinct.

Note

There's an additional way to run aggregation over a table. If a query contains table columns only inside aggregate functions, the `GROUP BY` clause can be omitted, and aggregation by an empty set of keys is assumed. Such queries always return exactly one row.

NULL Processing

For grouping, ClickHouse interprets `NULL` as a value, and `NULL==NULL`. It differs from `NULL` processing in most other contexts.

Here's an example to show what this means.

Assume you have this table:

x	y
1	2
2	NULL
3	2
3	3
3	NULL

The query `SELECT sum(x), y FROM t_null_big GROUP BY y` results in:

sum(x)	y
4	2
3	3
5	NULL

You can see that `GROUP BY` for `y = NULL` summed up `x`, as if `NULL` is this value.

If you pass several keys to `GROUP BY`, the result will give you all the combinations of the selection, as if `NULL` were a specific value.

WITH TOTALS Modifier

If the `WITH TOTALS` modifier is specified, another row will be calculated. This row will have key columns containing default values (zeros or empty lines), and columns of aggregate functions with the values calculated across all the rows (the “total” values).

This extra row is only produced in `JSON*`, `TabSeparated*`, and `Pretty*` formats, separately from the other rows:

- In `JSON*` formats, this row is output as a separate ‘totals’ field.
- In `TabSeparated*` formats, the row comes after the main result, preceded by an empty row (after the other data).
- In `Pretty*` formats, the row is output as a separate table after the main result.
- In the other formats it is not available.

`WITH TOTALS` can be run in different ways when `HAVING` is present. The behavior depends on the `totals_mode` setting.

Configuring Totals Processing

By default, `totals_mode = 'before_having'`. In this case, 'totals' is calculated across all rows, including the ones that don't pass through HAVING and `max_rows_to_group_by`.

The other alternatives include only the rows that pass through HAVING in 'totals', and behave differently with the setting `max_rows_to_group_by` and `group_by_overflow_mode = 'any'`.

`after_having_exclusive` – Don't include rows that didn't pass through `max_rows_to_group_by`. In other words, 'totals' will have less than or the same number of rows as it would if `max_rows_to_group_by` were omitted.

`after_having_inclusive` – Include all the rows that didn't pass through '`max_rows_to_group_by`' in 'totals'. In other words, 'totals' will have more than or the same number of rows as it would if `max_rows_to_group_by` were omitted.

`after_having_auto` – Count the number of rows that passed through HAVING. If it is more than a certain amount (by default, 50%), include all the rows that didn't pass through '`max_rows_to_group_by`' in 'totals'. Otherwise, do not include them.

`totals_auto_threshold` – By default, 0.5. The coefficient for `after_having_auto`.

If `max_rows_to_group_by` and `group_by_overflow_mode = 'any'` are not used, all variations of `after_having` are the same, and you can use any of them (for example, `after_having_auto`).

You can use `WITH TOTALS` in subqueries, including subqueries in the `JOIN` clause (in this case, the respective total values are combined).

Examples

Example:

```
SELECT
    count(),
    median(FetchTiming > 60 ? 60 : FetchTiming),
    count() - sum(Refresh)
FROM hits
```

However, in contrast to standard SQL, if the table doesn't have any rows (either there aren't any at all, or there aren't any after using WHERE to filter), an empty result is returned, and not the result from one of the rows containing the initial values of aggregate functions.

As opposed to MySQL (and conforming to standard SQL), you can't get some value of some column that is not in a key or aggregate function (except constant expressions). To work around this, you can use the 'any' aggregate function (get the first encountered value) or 'min/max'.

Example:

```
SELECT
    domainWithoutWWW(URL) AS domain,
    count(),
    any>Title) AS title -- getting the first occurred page header for each domain.
FROM hits
GROUP BY domain
```

For every different key value encountered, `GROUP BY` calculates a set of aggregate function values.

`GROUP BY` is not supported for array columns.

A constant can't be specified as arguments for aggregate functions. Example: `sum(1)`. Instead of this, you can get rid of the constant. Example: `count()`.

Implementation Details

Aggregation is one of the most important features of a column-oriented DBMS, and thus it's implementation is one of the most heavily optimized parts of ClickHouse. By default, aggregation is done in memory using a hash-table. It has 40+ specializations that are chosen automatically depending on "grouping key" data types.

GROUP BY in External Memory

You can enable dumping temporary data to the disk to restrict memory usage during `GROUP BY`.

The `max_bytes_before_external_group_by` setting determines the threshold RAM consumption for dumping `GROUP BY` temporary data to the file system. If set to 0 (the default), it is disabled.

When using `max_bytes_before_external_group_by`, we recommend that you set `max_memory_usage` about twice as high. This is necessary because there are two stages to aggregation: reading the data and forming intermediate data (1) and merging the intermediate data (2). Dumping data to the file system can only occur during stage 1. If the temporary data wasn't dumped, then stage 2 might require up to the same amount of memory as in stage 1.

For example, if `max_memory_usage` was set to 10000000000 and you want to use external aggregation, it makes sense to set `max_bytes_before_external_group_by` to 10000000000, and `max_memory_usage` to 20000000000. When external aggregation is triggered (if there was at least one dump of temporary data), maximum consumption of RAM is only slightly more than `max_bytes_before_external_group_by`.

With distributed query processing, external aggregation is performed on remote servers. In order for the requester server to use only a small amount of RAM, set `distributed_aggregation_memory_efficient` to 1.

When merging data flushed to the disk, as well as when merging results from remote servers when the `distributed_aggregation_memory_efficient` setting is enabled, consumes up to `1/256 * the_number_of_threads` from the total amount of RAM.

When external aggregation is enabled, if there was less than `max_bytes_before_external_group_by` of data (i.e. data was not flushed), the query runs just as fast as without external aggregation. If any temporary data was flushed, the run time will be several times longer (approximately three times).

If you have an `ORDER BY` with a `LIMIT` after `GROUP BY`, then the amount of used RAM depends on the amount of data in `LIMIT`, not in the whole table. But if the `ORDER BY` doesn't have `LIMIT`, don't forget to enable external sorting (`max_bytes_before_external_sort`).

HAVING Clause

Allows filtering the aggregation results produced by `GROUP BY`. It is similar to the `WHERE` clause, but the difference is that `WHERE` is performed before aggregation, while `HAVING` is performed after it.

It is possible to reference aggregation results from `SELECT` clause in `HAVING` clause by their alias.

Alternatively, `HAVING` clause can filter on results of additional aggregates that are not returned in query results.

Limitations

`HAVING` can't be used if aggregation is not performed. Use `WHERE` instead.

INTO OUTFILE Clause

Add the `INTO OUTFILE filename` clause (where `filename` is a string literal) to `SELECT` query to redirect its output to the specified file on the client side.

to the specified file on the client-side.

Implementation Details

- This functionality is available in the [command-line client](#) and [clickhouse-local](#). Thus a query sent via [HTTP interface](#) will fail.
- The query will fail if a file with the same filename already exists.
- The default [output format](#) is `TabSeparated` (like in the command-line client batch mode).

JOIN Clause

Join produces a new table by combining columns from one or multiple tables by using values common to each. It is a common operation in databases with SQL support, which corresponds to [relational algebra](#) join. The special case of one table join is often referred to as "self-join".

Syntax:

```
SELECT <expr_list>
FROM <left_table>
[GLOBAL] [ANY|ALL|ASOF] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI] JOIN <right_table>
(ON <expr_list>)|(USING <column_list>) ...
```

Expressions from `ON` clause and columns from `USING` clause are called "join keys". Unless otherwise stated, join produces a [Cartesian product](#) from rows with matching "join keys", which might produce results with much more rows than the source tables.

Supported Types of JOIN

All standard [SQL JOIN](#) types are supported:

- `INNER JOIN`, only matching rows are returned.
- `LEFT OUTER JOIN`, non-matching rows from left table are returned in addition to matching rows.
- `RIGHT OUTER JOIN`, non-matching rows from right table are returned in addition to matching rows.
- `FULL OUTER JOIN`, non-matching rows from both tables are returned in addition to matching rows.
- `CROSS JOIN`, produces cartesian product of whole tables, "join keys" are **not** specified.

`JOIN` without specified type implies `INNER`. Keyword `OUTER` can be safely omitted. Alternative syntax for `CROSS JOIN` is specifying multiple tables in [FROM clause](#) separated by commas.

Additional join types available in ClickHouse:

- `LEFT SEMI JOIN` and `RIGHT SEMI JOIN`, a whitelist on "join keys", without producing a cartesian product.
- `LEFT ANTI JOIN` and `RIGHT ANTI JOIN`, a blacklist on "join keys", without producing a cartesian product.

Strictness

Modifies how matching by "join keys" is performed

- `ALL` — The standard `JOIN` behavior in SQL as described above. The default.
- `ANY` — Partially (for opposite side of `LEFT` and `RIGHT`) or completely (for `INNER` and `FULL`) disables the cartesian product for standard `JOIN` types.
- `ASOF` — For joining sequences with a non-exact match. `ASOF JOIN` usage is described below.

Note

The default strictness value can be overridden using `join_default_strictness` setting.

ASOF JOIN Usage

ASOF JOIN is useful when you need to join records that have no exact match.

Tables for ASOF JOIN must have an ordered sequence column. This column cannot be alone in a table, and should be one of the data types: UInt32, UInt64, Float32, Float64, Date, and DateTime.

Syntax ASOF JOIN ... ON:

```
SELECT expressions_list
FROM table_1
ASOF LEFT JOIN table_2
ON equi_cond AND closest_match_cond
```

You can use any number of equality conditions and exactly one closest match condition. For example, SELECT count() FROM table_1 ASOF LEFT JOIN table_2 ON table_1.a == table_2.b AND table_2.t <= table_1.t.

Conditions supported for the closest match: >, >=, <, <=.

Syntax ASOF JOIN ... USING:

```
SELECT expressions_list
FROM table_1
ASOF JOIN table_2
USING (equi_column1, ... equi_columnN, asof_column)
```

ASOF JOIN uses equi_columnX for joining on equality and asof_column for joining on the closest match with the table_1.asof_column >= table_2.asof_column condition. The asof_column column always the last one in the USING clause.

For example, consider the following tables:

table_1			table_2		
event	ev_time	user_id	event	ev_time	user_id
event_1_1	12:00	42	event_2_1	11:59	42
event_1_2	13:00	42	event_2_2	12:30	42
...

ASOF JOIN can take the timestamp of a user event from table_1 and find an event in table_2 where the timestamp is closest to the timestamp of the event from table_1 corresponding to the closest match condition. Equal timestamp values are the closest if available. Here, the user_id column can be used for joining on equality and the ev_time column can be used for joining on the closest match. In our example, event_1_1 can be joined with event_2_1 and event_1_2 can be joined with event_2_3, but event_2_2 can't be joined.

Note

ASOF join is **not** supported in the **Join** table engine.

Distributed Join

There are two ways to execute join involving distributed tables:

- When using a normal JOIN, the query is sent to remote servers. Subqueries are run on each of them in

order to make the right table, and the join is performed with this table. In other words, the right table is formed on each server separately.

- When using `GLOBAL ... JOIN`, first the requestor server runs a subquery to calculate the right table. This temporary table is passed to each remote server, and queries are run on them using the temporary data that was transmitted.

Be careful when using `GLOBAL`. For more information, see the [Distributed subqueries](#) section.

Usage Recommendations

Processing of Empty or NULL Cells

While joining tables, the empty cells may appear. The setting `join_use_nulls` define how ClickHouse fills these cells.

If the `JOIN` keys are [Nullable](#) fields, the rows where at least one of the keys has the value [NULL](#) are not joined.

Syntax

The columns specified in `USING` must have the same names in both subqueries, and the other columns must be named differently. You can use aliases to change the names of columns in subqueries.

The `USING` clause specifies one or more columns to join, which establishes the equality of these columns. The list of columns is set without brackets. More complex join conditions are not supported.

Syntax Limitations

For multiple `JOIN` clauses in a single `SELECT` query:

- Taking all the columns via `*` is available only if tables are joined, not subqueries.
- The `PREWHERE` clause is not available.

For `ON`, `WHERE`, and `GROUP BY` clauses:

- Arbitrary expressions cannot be used in `ON`, `WHERE`, and `GROUP BY` clauses, but you can define an expression in a `SELECT` clause and then use it in these clauses via an alias.

Performance

When running a `JOIN`, there is no optimization of the order of execution in relation to other stages of the query. The join (a search in the right table) is run before filtering in `WHERE` and before aggregation.

Each time a query is run with the same `JOIN`, the subquery is run again because the result is not cached. To avoid this, use the special [Join](#) table engine, which is a prepared array for joining that is always in RAM.

In some cases, it is more efficient to use `IN` instead of `JOIN`.

If you need a `JOIN` for joining with dimension tables (these are relatively small tables that contain dimension properties, such as names for advertising campaigns), a `JOIN` might not be very convenient due to the fact that the right table is re-accessed for every query. For such cases, there is an “external dictionaries” feature that you should use instead of `JOIN`. For more information, see the [External dictionaries](#) section.

Memory Limitations

By default, ClickHouse uses the [hash join](#) algorithm. ClickHouse takes the `<right_table>` and creates a hash table for it in RAM. After some threshold of memory consumption, ClickHouse falls back to merge join algorithm.

If you need to restrict join operation memory consumption use the following settings:

- `max_rows_in_join` — Limits number of rows in the hash table.
- `max_bytes_in_join` — Limits size of the hash table.

When any of these limits is reached, ClickHouse acts as the [join_overflow_mode](#) setting instructs.

Examples

Example:

```
SELECT
    CounterID,
    hits,
    visits
FROM
(
    SELECT
        CounterID,
        count() AS hits
    FROM test.hits
    GROUP BY CounterID
) ANY LEFT JOIN
(
    SELECT
        CounterID,
        sum(Sign) AS visits
    FROM test.visits
    GROUP BY CounterID
) USING CounterID
ORDER BY hits DESC
LIMIT 10
```

CounterID	hits	visits
1143050	523264	13665
731962	475698	102716
722545	337212	108187
722889	252197	10547
2237260	196036	9522
23057320	147211	7689
722818	90109	17847
48221	85379	4652
19762435	77807	7026
722884	77492	11056

LIMIT Clause

`LIMIT m` allows to select the first `m` rows from the result.

`LIMIT n, m` allows to select the `m` rows from the result after skipping the first `n` rows. The `LIMIT m OFFSET n` syntax is equivalent.

`n` and `m` must be non-negative integers.

If there is no `ORDER BY` clause that explicitly sorts results, the choice of rows for the result may be arbitrary and non-deterministic.

LIMIT BY Clause

A query with the `LIMIT n BY expressions` clause selects the first `n` rows for each distinct value of `expressions`.

The key for `LIMIT BY` can contain any number of **expressions**.

ClickHouse supports the following syntax variants:

- `LIMIT [offset_value,]n BY` expressions
- `LIMIT n OFFSET offset_value BY` expressions

During query processing, ClickHouse selects data ordered by sorting key. The sorting key is set explicitly using an `ORDER BY` clause or implicitly as a property of the table engine. Then ClickHouse applies `LIMIT n BY` expressions and returns the first `n` rows for each distinct combination of expressions. If `OFFSET` is specified, then for each data block that belongs to a distinct combination of expressions, ClickHouse skips `offset_value` number of rows from the beginning of the block and returns a maximum of `n` rows as a result. If `offset_value` is bigger than the number of rows in the data block, ClickHouse returns zero rows from the block.

Note

`LIMIT BY` is not related to `LIMIT`. They can both be used in the same query.

Examples

Sample table:

```
CREATE TABLE limit_by(id Int, val Int) ENGINE = Memory;
INSERT INTO limit_by VALUES (1, 10), (1, 11), (1, 12), (2, 20), (2, 21);
```

Queries:

```
SELECT * FROM limit_by ORDER BY id, val LIMIT 2 BY id
```

id	val
1	10
1	11
2	20
2	21

```
SELECT * FROM limit_by ORDER BY id, val LIMIT 1, 2 BY id
```

id	val
1	11
1	12
2	21

The `SELECT * FROM limit_by ORDER BY id, val LIMIT 2 OFFSET 1 BY id` query returns the same result.

The following query returns the top 5 referrers for each `domain, device_type` pair with a maximum of 100 rows in total (`LIMIT n BY + LIMIT`).

```
SELECT
    domainWithoutWWW(URL) AS domain,
    domainWithoutWWW(REFERRER_URL) AS referrer,
    device_type,
```

```
count() cnt
FROM hits
GROUP BY domain, referrer, device_type
ORDER BY cnt DESC
LIMIT 5 BY domain, device_type
LIMIT 100
```

ORDER BY Clause

The `ORDER BY` clause contains a list of expressions, which can each be attributed with `DESC` (descending) or `ASC` (ascending) modifier which determine the sorting direction. If the direction is not specified, `ASC` is assumed, so it's usually omitted. The sorting direction applies to a single expression, not to the entire list.

Example: `ORDER BY Visits DESC, SearchPhrase`

Rows that have identical values for the list of sorting expressions are output in an arbitrary order, which can also be non-deterministic (different each time).

If the `ORDER BY` clause is omitted, the order of the rows is also undefined, and may be non-deterministic as well.

Sorting of Special Values

There are two approaches to `Nan` and `NULL` sorting order:

- By default or with the `NULLS LAST` modifier: first the values, then `Nan`, then `NULL`.
- With the `NULLS FIRST` modifier: first `NULL`, then `Nan`, then other values.

Example

For the table

x	y
1	NULL
2	2
1	nan
2	2
3	4
5	6
6	nan
7	NULL
6	7
8	9

Run the query `SELECT * FROM t_null_nan ORDER BY y NULLS FIRST` to get:

x	y
1	NULL
7	NULL
1	nan
6	nan
2	2
2	2
3	4
5	6
6	7
8	9

When floating point numbers are sorted, NaNs are separate from the other values. Regardless of the sorting order, NaNs come at the end. In other words, for ascending sorting they are placed as if they are larger than all the other numbers, while for descending sorting they are placed as if they are smaller than the rest.

Collation Support

For sorting by String values, you can specify collation (comparison). Example: `ORDER BY SearchPhrase COLLATE 'tr'` - for sorting by keyword in ascending order, using the Turkish alphabet, case insensitive, assuming that strings are UTF-8 encoded. `COLLATE` can be specified or not for each expression in `ORDER BY` independently. If `ASC` or `DESC` is specified, `COLLATE` is specified after it. When using `COLLATE`, sorting is always case-insensitive.

We only recommend using `COLLATE` for final sorting of a small number of rows, since sorting with `COLLATE` is less efficient than normal sorting by bytes.

Implementation Details

Less RAM is used if a small enough `LIMIT` is specified in addition to `ORDER BY`. Otherwise, the amount of memory spent is proportional to the volume of data for sorting. For distributed query processing, if `GROUP BY` is omitted, sorting is partially done on remote servers, and the results are merged on the requestor server. This means that for distributed sorting, the volume of data to sort can be greater than the amount of memory on a single server.

If there is not enough RAM, it is possible to perform sorting in external memory (creating temporary files on a disk). Use the setting `max_bytes_before_external_sort` for this purpose. If it is set to 0 (the default), external sorting is disabled. If it is enabled, when the volume of data to sort reaches the specified number of bytes, the collected data is sorted and dumped into a temporary file. After all data is read, all the sorted files are merged and the results are output. Files are written to the `/var/lib/clickhouse/tmp/` directory in the config (by default, but you can use the `tmp_path` parameter to change this setting).

Running a query may use more memory than `max_bytes_before_external_sort`. For this reason, this setting must have a value significantly smaller than `max_memory_usage`. As an example, if your server has 128 GB of RAM and you need to run a single query, set `max_memory_usage` to 100 GB, and `max_bytes_before_external_sort` to 80 GB.

External sorting works much less effectively than sorting in RAM.

PREWHERE Clause

Prewhere is an optimization to apply filtering more efficiently. It is enabled by default even if `PREWHERE` clause is not specified explicitly. It works by automatically moving part of `WHERE` condition to prewhere stage. The role of `PREWHERE` clause is only to control this optimization if you think that you know how to do it better than it happens by default.

With prewhere optimization, at first only the columns necessary for executing prewhere expression are read. Then the other columns are read that are needed for running the rest of the query, but only those blocks where the prewhere expression is "true" at least for some rows. If there are a lot of blocks where prewhere expression is "false" for all rows and prewhere needs less columns than other parts of query, this often allows to read a lot less data from disk for query execution.

Controlling Prewhere Manually

The clause has the same meaning as the `WHERE` clause. The difference is in which data is read from the table. When manually controlling `PREWHERE` for filtration conditions that are used by a minority of the columns in the query, but that provide strong data filtration. This reduces the volume of data to read.

A query may simultaneously specify `PREWHERE` and `WHERE`. In this case, `PREWHERE` precedes `WHERE`.

If the `optimize_move_to_prewhere` setting is set to 0, heuristics to automatically move parts of expressions from `WHERE` to `PREWHERE` are disabled.

Limitations

`PREWHERE` is only supported by tables from the `*MergeTree` family.

SAMPLE Clause

The `SAMPLE` clause allows for approximated `SELECT` query processing.

When data sampling is enabled, the query is not performed on all the data, but only on a certain fraction of data (sample). For example, if you need to calculate statistics for all the visits, it is enough to execute the query on the 1/10 fraction of all the visits and then multiply the result by 10.

Approximated query processing can be useful in the following cases:

- When you have strict timing requirements (like $<100\text{ms}$) but you can't justify the cost of additional hardware resources to meet them.
- When your raw data is not accurate, so approximation doesn't noticeably degrade the quality.
- Business requirements target approximate results (for cost-effectiveness, or to market exact results to premium users).

Note

You can only use sampling with the tables in the `MergeTree` family, and only if the sampling expression was specified during table creation (see [MergeTree engine](#)).

The features of data sampling are listed below:

- Data sampling is a deterministic mechanism. The result of the same `SELECT .. SAMPLE` query is always the same.
- Sampling works consistently for different tables. For tables with a single sampling key, a sample with the same coefficient always selects the same subset of possible data. For example, a sample of user IDs takes rows with the same subset of all the possible user IDs from different tables. This means that you can use the sample in subqueries in the `IN` clause. Also, you can join samples using the `JOIN` clause.
- Sampling allows reading less data from a disk. Note that you must specify the sampling key correctly. For more information, see [Creating a MergeTree Table](#).

For the `SAMPLE` clause the following syntax is supported:

SAMPLE Clause Syntax	Description
<code>SAMPLE k</code>	Here <code>k</code> is the number from 0 to 1.

The query is executed on `k` fraction of data. For example, `SAMPLE 0.1` runs the query on 10% of data. [Read more](#) `SAMPLE n` Here `n` is a sufficiently large integer. The query is executed on a sample of at least `n` rows (but not significantly more than this). For example, `SAMPLE 10000000` runs the query on a minimum of 10,000,000 rows. [Read more](#) `SAMPLE k OFFSET m` Here `k` and `m` are the numbers from 0 to 1. The query is executed on a sample of `k` fraction of the data. The data used for the sample is offset by `m` fraction. [Read more](#)

SAMPLE K

Here `k` is the number from 0 to 1 (both fractional and decimal notations are supported). For example `SAMPLE`

Here k is the number from 0 to 1 (both fractional and decimal notations are supported). For example, SAMPLE 1/2 or SAMPLE 0.5.

In a SAMPLE k clause, the sample is taken from the k fraction of data. The example is shown below:

```
SELECT
    Title,
    count() * 10 AS PageViews
FROM hits_distributed
SAMPLE 0.1
WHERE
    CounterID = 34
GROUP BY Title
ORDER BY PageViews DESC LIMIT 1000
```

In this example, the query is executed on a sample from 0.1 (10%) of data. Values of aggregate functions are not corrected automatically, so to get an approximate result, the value `count()` is manually multiplied by 10.

SAMPLE N

Here n is a sufficiently large integer. For example, SAMPLE 10000000.

In this case, the query is executed on a sample of at least n rows (but not significantly more than this). For example, SAMPLE 10000000 runs the query on a minimum of 10,000,000 rows.

Since the minimum unit for data reading is one granule (its size is set by the `index_granularity` setting), it makes sense to set a sample that is much larger than the size of the granule.

When using the SAMPLE n clause, you don't know which relative percent of data was processed. So you don't know the coefficient the aggregate functions should be multiplied by. Use the `_sample_factor` virtual column to get the approximate result.

The `_sample_factor` column contains relative coefficients that are calculated dynamically. This column is created automatically when you `create` a table with the specified sampling key. The usage examples of the `_sample_factor` column are shown below.

Let's consider the table `visits`, which contains the statistics about site visits. The first example shows how to calculate the number of page views:

```
SELECT sum(PageViews * _sample_factor)
FROM visits
SAMPLE 10000000
```

The next example shows how to calculate the total number of visits:

```
SELECT sum(_sample_factor)
FROM visits
SAMPLE 10000000
```

The example below shows how to calculate the average session duration. Note that you don't need to use the relative coefficient to calculate the average values.

```
SELECT avg(Duration)
FROM visits
SAMPLE 10000000
```

SAMPLE K OFFSET M

Here `k` and `m` are numbers from 0 to 1. Examples are shown below.

Example 1

```
SAMPLE 1/10
```

In this example, the sample is 1/10th of all data:

```
[+-----]
```

Example 2

```
SAMPLE 1/10 OFFSET 1/2
```

Here, a sample of 10% is taken from the second half of the data.

```
[-----+-----]
```

UNION ALL Clause

You can use `UNION ALL` to combine any number of `SELECT` queries by extending their results. Example:

```
SELECT CounterID, 1 AS table,.toInt64(count()) AS c
  FROM test.hits
  GROUP BY CounterID

UNION ALL

SELECT CounterID, 2 AS table, sum(Sign) AS c
  FROM test.visits
  GROUP BY CounterID
  HAVING c > 0
```

Result columns are matched by their index (order inside `SELECT`). If column names do not match, names for the final result are taken from the first query.

Type casting is performed for unions. For example, if two queries being combined have the same field with non-`Nullable` and `Nullable` types from a compatible type, the resulting `UNION ALL` has a `Nullable` type field.

Queries that are parts of `UNION ALL` can't be enclosed in round brackets. `ORDER BY` and `LIMIT` are applied to separate queries, not to the final result. If you need to apply a conversion to the final result, you can put all the queries with `UNION ALL` in a subquery in the `FROM` clause.

Limitations

Only `UNION ALL` is supported. The regular `UNION` (`UNION DISTINCT`) is not supported. If you need `UNION DISTINCT`, you can write `SELECT DISTINCT` from a subquery containing `UNION ALL`.

Implementation Details

Queries that are parts of `UNION ALL` can be run simultaneously, and their results can be mixed together.

WHERE Clause

`WHERE` clause allows to filter the data that is coming from `FROM` clause of `SELECT`.

If there is a `WHERE` clause, it must contain an expression with the `UInt8` type. This is usually an expression with comparison and logical operators. Rows where this expression evaluates to 0 are excluded from further transformations or result.

`WHERE` expression is evaluated on the ability to use indexes and partition pruning, if the underlying table engine supports that.

Note

There's a filtering optimization called **prewhere**.

WITH Clause

This section provides support for Common Table Expressions (**CTE**), so the results of `WITH` clause can be used in the rest of `SELECT` query.

Limitations

1. Recursive queries are not supported.
2. When subquery is used inside WITH section, its result should be scalar with exactly one row.
3. Expression's results are not available in subqueries.

Examples

Example 1: Using constant expression as “variable”

```
WITH '2019-08-01 15:23:00' as ts_upper_bound
SELECT *
FROM hits
WHERE
    EventDate = toDate(ts_upper_bound) AND
    EventTime <= ts_upper_bound
```

Example 2: Evicting `sum(bytes)` expression result from `SELECT` clause column list

```
WITH sum(bytes) as s
SELECT
    formatReadableSize(s),
    table
FROM system.parts
GROUP BY table
ORDER BY s
```

Example 3: Using results of scalar subquery

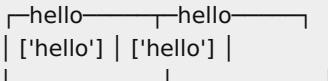
```
/* this example would return TOP 10 of most huge tables */
WITH
(
    SELECT sum(bytes)
    FROM system.parts
    WHERE active
) AS total_disk_usage
SELECT
    (sum(bytes) / total_disk_usage) * 100 AS table_disk_usage,
    table
```

```
FROM system.parts
GROUP BY table
ORDER BY table_disk_usage DESC
LIMIT 10
```

Example 4: Re-using expression in subquery

As a workaround for current limitation for expression usage in subqueries, you may duplicate it.

```
WITH ['hello'] AS hello
SELECT
    hello,
    *
FROM
(
    WITH ['hello'] AS hello
    SELECT hello
)
```



ALTER

该 `ALTER` 查询仅支持 `*MergeTree` 表，以及 `Merge` 和 `Distributed`. 查询有几个变体。

列操作

更改表结构。

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

在查询中，指定一个或多个逗号分隔操作的列表。

每个操作都是对列的操作。

支持以下操作：

- **ADD COLUMN** — Adds a new column to the table.
- **DROP COLUMN** — Deletes the column.
- **CLEAR COLUMN** — Resets column values.
- **COMMENT COLUMN** — Adds a text comment to the column.
- **MODIFY COLUMN** — Changes column's type, default expression and TTL.

下面详细描述这些动作。

ADD COLUMN

```
ADD COLUMN [IF NOT EXISTS] name [type] [default_expr] [codec] [AFTER name_after]
```

将一个新列添加到表中，并指定 `name`, `type`, `codec` 和 `default_expr` (请参阅部分 [默认表达式](#)).

如果 `IF NOT EXISTS` 如果列已经存在，则查询不会返回错误。如果您指定 `AFTER name_after` (另一列的名称)，该列被添加在表列表中指定的一列之后。否则，该列将添加到表的末尾。请注意，没有办法将列添加到表的开头。为了一系列的行

动, `name_after` 可以是在以前的操作之一中添加的列的名称。

添加列只是更改表结构, 而不对数据执行任何操作。数据不会出现在磁盘上后 `ALTER`. 如果从表中读取某一列的数据缺失, 则将使用默认值填充该列 (如果存在默认表达式, 则执行默认表达式, 或使用零或空字符串)。合并数据部分后, 该列将出现在磁盘上 (请参阅 [MergeTree](#))。

这种方法使我们能够完成 `ALTER` 即时查询, 不增加旧数据量。

示例:

```
ALTER TABLE visits ADD COLUMN browser String AFTER user_id
```

DROP COLUMN

```
DROP COLUMN [IF EXISTS] name
```

删除具有名称的列 `name`. 如果 `IF EXISTS` 如果指定了子句, 如果该列不存在, 则查询不会返回错误。

从文件系统中删除数据。由于这将删除整个文件, 查询几乎立即完成。

示例:

```
ALTER TABLE visits DROP COLUMN browser
```

CLEAR COLUMN

```
CLEAR COLUMN [IF EXISTS] name IN PARTITION partition_name
```

重置指定分区的列中的所有数据。了解有关设置分区名称的详细信息 [如何指定分区表达式](#).

如果 `IF EXISTS` 如果指定了子句, 如果该列不存在, 则查询不会返回错误。

示例:

```
ALTER TABLE visits CLEAR COLUMN browser IN PARTITION tuple()
```

COMMENT COLUMN

```
COMMENT COLUMN [IF EXISTS] name 'comment'
```

向列添加注释。如果 `IF EXISTS` 如果指定了子句, 如果该列不存在, 则查询不会返回错误。

每列可以有一个注释。如果列的注释已存在, 则新注释将复盖以前的注释。

注释存储在 `comment_expression` 由返回的列 `DESCRIBE TABLE` 查询。

示例:

```
ALTER TABLE visits COMMENT COLUMN browser 'The table shows the browser used for accessing the site.'
```

MODIFY COLUMN

```
MODIFY COLUMN [IF EXISTS] name [type] [default_expr] [TTL]
```

此查询更改以 `name` 列属性：

- 类型
- 默认表达式
- TTL

For examples of columns TTL modifying, see [Column TTL]
([./engines/table_engines/mergetree_family/mergetree.md#mergetree-column-ttl](#)).

如果 `IF EXISTS` 如果指定了子句，如果该列不存在，则查询不会返回错误。

更改类型时，值将被转换为 `toType` 函数被应用到它们。如果仅更改默认表达式，则查询不会执行任何复杂的操作，并且几乎立即完成。

示例：

```
ALTER TABLE visits MODIFY COLUMN browser Array(String)
```

Changing the column type is the only complex action – it changes the contents of files with data. For large tables, this may take a long time.

有几个处理阶段：

- 准备具有修改数据的临时（新）文件。
- 重命名旧文件。
- 将临时（新）文件重命名为旧名称。
- 删除旧文件。

只有第一阶段需要时间。如果在此阶段出现故障，则不会更改数据。

如果在其中一个连续阶段中出现故障，可以手动恢复数据。例外情况是，如果旧文件从文件系统中删除，但新文件的数据没有写入磁盘并丢失。

该 `ALTER` 复制更改列的查询。这些指令保存在 ZooKeeper 中，然后每个副本应用它们。全部 `ALTER` 查询以相同的顺序运行。查询等待对其他副本完成适当的操作。但是，更改复制表中的列的查询可能会中断，并且所有操作都将异步执行。

更改查询限制

该 `ALTER query` 允许您在嵌套数据结构中创建和删除单独的元素（列），但不能创建整个嵌套数据结构。要添加嵌套数据结构，可以添加名称如下的列 `name.nested_name` 和类型 `Array(T)`。嵌套数据结构等效于名称在点之前具有相同前缀的多个数组列。

不支持删除主键或采样键中的列（在主键中使用的列 `ENGINE` 表达式）。只有在此更改不会导致数据被修改时，才可以更改主键中包含的列的类型（例如，允许您向枚举添加值或更改类型 `DateTime` 到 `UInt32`）。

如果 `ALTER` 查询不足以使您需要的表更改，您可以创建一个新的表，使用 `INSERT SELECT` 查询，然后使用切换表 `RENAME` 查询并删除旧表。您可以使用 `ツ環板-ヨツ嘉ツツ徳` 作为替代 `INSERT SELECT` 查询。

该 `ALTER` 查询阻止对表的所有读取和写入。换句话说，如果长 `SELECT` 正在运行的时间 `ALTER` 查询，该 `ALTER` 查询将等待它完成。同时，对同一个表的所有新查询将等待 `ALTER` 正在运行。

对于本身不存储数据的表（例如 `Merge` 和 `Distributed`），`ALTER` 只是改变了表结构，并且不改变从属表的结构。例如，当运行 `ALTER` 时 `Distributed` 表，你还需要运行 `ALTER` 对于所有远程服务器上的表。

使用键表达式进行操作

支持以下命令：

```
MODIFY ORDER BY new expression
```

它只适用于在表 `MergeTree` 家庭（包括
复制 表）。该命令更改
排序键 表
到 `new_expression`（表达式或表达式元组）。主键保持不变。

该命令是轻量级的，因为它只更改元数据。要保持该数据部分的属性
行按排序键表达式排序您不能添加包含现有列的表达式
到排序键（仅由列添加 `ADD COLUMN` 命令在同一个 `ALTER` 查询）。

使用数据跳过索引进行操作

它只适用于在表 `*MergeTree` 家庭（包括
复制 表）。以下操作

可用：

- `ALTER TABLE [db].name ADD INDEX name expression TYPE type GRANULARITY value AFTER name [AFTER name2]`-将索引描述添加到表元数据。
- `ALTER TABLE [db].name DROP INDEX name` -从表元数据中删除索引描述并从磁盘中删除索引文件。

这些命令是轻量级的，因为它们只更改元数据或删除文件。

此外，它们被复制（通过ZooKeeper同步索引元数据）。

使用约束进行操作

查看更多 [制约因素](#)

可以使用以下语法添加或删除约束：

```
ALTER TABLE [db].name ADD CONSTRAINT constraint_name CHECK expression;
ALTER TABLE [db].name DROP CONSTRAINT constraint_name;
```

查询将从表中添加或删除有关约束的元数据，以便立即处理它们。

约束检查 不会被执行在现有数据上，如果它被添加。

复制表上的所有更改都广播到ZooKeeper，因此将应用于其他副本。

操作与分区和零件

下面的操作与 [分区](#) 可用：

- **DETACH PARTITION** – Moves a partition to the `detached` 目录和忘记它。
- **DROP PARTITION** – Deletes a partition.
- **ATTACH PART|PARTITION** – Adds a part or partition from the `detached` 目录到表。
- **ATTACH PARTITION FROM** – Copies the data partition from one table to another and adds.
- **REPLACE PARTITION** -将数据分区从一个表复制到另一个表并替换。
- **MOVE PARTITION TO TABLE**(#alter_move_to_table-partition)-将数据分区从一个表移动到另一个表。
- **CLEAR COLUMN IN PARTITION** -重置分区中指定列的值。
- **CLEAR INDEX IN PARTITION** -重置分区中指定的二级索引。
- **FREEZE PARTITION** – Creates a backup of a partition.
- **FETCH PARTITION** – Downloads a partition from another server.
- **MOVE PARTITION|PART** – Move partition/data part to another disk or volume.

DETACH PARTITION

```
ALTER TABLE table_name DETACH PARTITION partition_expr
```

将指定分区的所有数据移动到 `detached` 目录。服务器会忘记分离的数据分区，就好像它不存在一样。服务器不会知道这个数据，直到你做 **ATTACH** 查询。

示例：

```
ALTER TABLE visits DETACH PARTITION 201901
```

阅读有关在一节中设置分区表达式的信息 [如何指定分区表达式](#)。

执行查询后，您可以对查询中的数据进行任何操作 `detached` directory — delete it from the file system, or just leave it.

This query is replicated – it moves the data to the `detached` 所有副本上的目录。请注意，您只能对领导副本执行此查询。要确定副本是否为领导者，请执行 `SELECT` 查询到 [系统。副本桌子](#) 或者，它更容易使 `DETACH` 对所有副本进行查询-除了领导副本之外，所有副本都会引发异常。

DROP PARTITION

```
ALTER TABLE table_name DROP PARTITION partition_expr
```

从表中删除指定的分区。此查询将分区标记为非活动分区，并在大约10分钟内完全删除数据。

阅读有关在一节中设置分区表达式的信息 [如何指定分区表达式](#)。

The query is replicated – it deletes data on all replicas.

DROP DETACHED PARTITION|PART

```
ALTER TABLE table_name DROP DETACHED PARTITION|PART partition_expr
```

从中删除指定分区的指定部分或所有部分 `detached`。

了解有关在一节中设置分区表达式的详细信息 [如何指定分区表达式](#)。

ATTACH PARTITION|PART

```
ALTER TABLE table_name ATTACH PARTITION|PART partition_expr
```

将数据从 `detached` 目录。可以为整个分区或单独的部分添加数据。例：

```
ALTER TABLE visits ATTACH PARTITION 201901;
ALTER TABLE visits ATTACH PART 201901_2_2_0;
```

了解有关在一节中设置分区表达式的详细信息 [如何指定分区表达式](#)。

此查询被复制。副本发起程序检查是否有数据在 `detached` 目录。如果数据存在，则查询将检查其完整性。如果一切正确，则查询将数据添加到表中。所有其他副本都从副本发起程序下载数据。

所以你可以把数据到 `detached` 在一个副本上的目录，并使用 `ALTER ... ATTACH` 查询以将其添加到所有副本上的表中。

ATTACH PARTITION FROM

```
ALTER TABLE table2 ATTACH PARTITION partition_expr FROM table1
```

此查询将数据分区从 `table1` 到 `table2` 将数据添加到存在 `table2`。请注意，数据不会从中删除 `table1`。

要使查询成功运行，必须满足以下条件：

- 两个表必须具有相同的结构。
- 两个表必须具有相同的分区键。

REPLACE PARTITION

```
ALTER TABLE table2 REPLACE PARTITION partition_expr FROM table1
```

此查询将数据分区从 `table1` 到 `table2` 并替换在现有的分区 `table2`. 请注意，数据不会从中删除 `table1`.

要使查询成功运行，必须满足以下条件:

- 两个表必须具有相同的结构。
- 两个表必须具有相同的分区键。

MOVE PARTITION TO TABLE

```
ALTER TABLE table_source MOVE PARTITION partition_expr TO TABLE table_dest
```

此查询将数据分区从 `table_source` 到 `table_dest` 删除数据 `table_source`.

要使查询成功运行，必须满足以下条件:

- 两个表必须具有相同的结构。
- 两个表必须具有相同的分区键。
- 两个表必须是相同的引擎系列。 (已复制或未复制)
- 两个表必须具有相同的存储策略。

CLEAR COLUMN IN PARTITION

```
ALTER TABLE table_name CLEAR COLUMN column_name IN PARTITION partition_expr
```

重置分区中指定列中的所有值。如果 `DEFAULT` 创建表时确定了子句，此查询将列值设置为指定的默认值。

示例:

```
ALTER TABLE visits CLEAR COLUMN hour in PARTITION 201902
```

FREEZE PARTITION

```
ALTER TABLE table_name FREEZE [PARTITION partition_expr]
```

此查询创建指定分区的本地备份。如果 `PARTITION` 子句被省略，查询一次创建所有分区的备份。

注

在不停止服务器的情况下执行整个备份过程。

请注意，对于旧式表，您可以指定分区名称的前缀（例如，'2019'）-然后查询为所有相应的分区创建备份。阅读有关在一节中设置分区表达式的信息 [如何指定分区表达式](#).

在执行时，对于数据快照，查询将创建指向表数据的硬链接。硬链接被放置在目录中 `/var/lib/clickhouse/shadow/N/...`，哪里:

- `/var/lib/clickhouse/` 是配置中指定的工作ClickHouse目录。
- `N` 是备份的增量编号。

注

如果您使用 `用于在表中存储数据的一组磁盘`，该 `shadow/N` 目录出现在每个磁盘上，存储由匹配的数据部分 `PARTITION` 表达。

在备份内部创建的目录结构与在备份内部创建的目录结构相同 `/var/lib/clickhouse/`。查询执行 `'chmod'` 对于所有文件，禁止写入它们。

创建备份后，您可以从以下位置复制数据 `/var/lib/clickhouse/shadow/` 然后将其从本地服务器中删除。请注意，`ALTER t FREEZE PARTITION` 不复制查询。它仅在本地服务器上创建本地备份。

查询几乎立即创建备份（但首先它会等待对相应表的当前查询完成运行）。

`ALTER TABLE t FREEZE PARTITION` 仅复制数据，而不复制表元数据。若要备份表元数据，请复制该文件 `/var/lib/clickhouse/metadata/database/table.sql`

要从备份还原数据，请执行以下操作：

1. 如果表不存在，则创建该表。要查看查询，请使用。`sql`文件（替换 `ATTACH` 在它与 `CREATE`）。
2. 从复制数据 `data/database/table/` 目录内的备份到 `/var/lib/clickhouse/data/database/table/detached/` 目录。
3. 快跑 `ALTER TABLE t ATTACH PARTITION` 将数据添加到表的查询。

从备份还原不需要停止服务器。

有关备份和还原数据的详细信息，请参阅 [数据备份](#) 科。

CLEAR INDEX IN PARTITION

```
ALTER TABLE table_name CLEAR INDEX index_name IN PARTITION partition_expr
```

查询的工作原理类似于 `CLEAR COLUMN`，但它重置索引而不是列数据。

FETCH PARTITION

```
ALTER TABLE table_name FETCH PARTITION partition_expr FROM 'path-in-zookeeper'
```

从另一台服务器下载分区。此查询仅适用于复制的表。

查询执行以下操作：

1. 从指定的分片下载分区。在 `'path-in-zookeeper'` 您必须在ZooKeeper中指定分片的路径。
2. 然后查询将下载的数据放到 `detached` 的目录 `table_name` 桌子 使用 `ATTACH PARTITION|PART` 查询将数据添加到表中。

例如：

```
ALTER TABLE users FETCH PARTITION 201902 FROM '/clickhouse/tables/01-01/visits';
ALTER TABLE users ATTACH PARTITION 201902;
```

请注意：

- 该 `ALTER ... FETCH PARTITION` 查询不被复制。它将分区放置在 `detached` 仅在本地服务器上的目录。
- 该 `ALTER TABLE ... ATTACH` 复制查询。它将数据添加到所有副本。数据被添加到从副本之一 `detached` 目录，以及其他-从相邻的副本。

在下载之前，系统会检查分区是否存在并且表结构匹配。从正常副本中自动选择最合适的副本。

虽然查询被调用 `ALTER TABLE`，它不会更改表结构，并且不会立即更改表中可用的数据。

MOVE PARTITION|PART

将分区或数据部分移动到另一个卷或磁盘 MergeTree-发动机表。看 [使用多个块设备进行数据存储](#).

```
ALTER TABLE table_name MOVE PARTITION|PART partition_expr TO DISK|VOLUME 'disk_name'
```

该 `ALTER TABLE t MOVE` 查询:

- 不复制，因为不同的副本可能具有不同的存储策略。
- 如果未配置指定的磁盘或卷，则返回错误。如果无法应用存储策略中指定的数据移动条件，Query还会返回错误。
- 可以在返回错误的情况下，当要移动的数据已经被后台进程移动时，并发 `ALTER TABLE t MOVE` 查询或作为后台数据合并的结果。在这种情况下，用户不应该执行任何其他操作。

示例:

```
ALTER TABLE hits MOVE PART '20190301_14343_16206_438' TO VOLUME 'slow'  
ALTER TABLE hits MOVE PARTITION '2019-09-01' TO DISK 'fast_ssd'
```

如何设置分区表达式

您可以在以下内容中指定分区表达式 `ALTER ... PARTITION` 以不同方式查询:

- 作为从值 `partition` 列 `system.parts` 桌子 例如, `ALTER TABLE visits DETACH PARTITION 201901`.
- 作为来自表列的表达式。支持常量和常量表达式。例如, `ALTER TABLE visits DETACH PARTITION toYYYYMM(toDate('2019-01-25'))`.
- 使用分区ID。分区ID是用作文件系统和ZooKeeper中分区名称的分区的字符串标识符（如果可能的话，人类可读）。分区ID必须在指定 `PARTITION ID` 子句，用单引号。例如, `ALTER TABLE visits DETACH PARTITION ID '201901'`.
- 在 `ALTER ATTACH PART` 和 `DROP DETACHED PART` 查询时，要指定部件的名称，请将字符串文字与来自 `name` 列 系统。`detached_parts` 桌子 例如, `ALTER TABLE visits ATTACH PART '201901_1_1_0'`.

指定分区时引号的使用取决于分区表达式的类型。例如，对于 `String` 类型，你必须在引号中指定其名称 ('). 为 `Date` 和 `Int*` 类型不需要引号。

对于旧式表，您可以将分区指定为数字 `201901` 或者一个字符串 `'201901'`. 对于类型，新样式表的语法更严格（类似于值输入格式的解析器）。

上述所有规则也适用于 `OPTIMIZE` 查询。如果在优化非分区表时需要指定唯一的分区，请设置表达式 `PARTITION tuple()`. 例如:

```
OPTIMIZE TABLE table_not_partitioned PARTITION tuple() FINAL;
```

的例子 `ALTER ... PARTITION` 查询在测试中演示 `00502_custom_partitioning_local` 和 `00502_custom_partitioning_replicated_zookeeper`.

使用表TTL进行操作

你可以改变 表TTL 请填写以下表格:

```
ALTER TABLE table-name MODIFY TTL ttl-expression
```

ALTER查询的同步性

对于不可复制的表，所有 `ALTER` 查询是同步执行的。对于可复制的表，查询仅添加相应操作的说明 ZooKeeper，并尽快执行操作本身。但是，查询可以等待在所有副本上完成这些操作。

为 `ALTER ... ATTACH|DETACH|DROP` 查询，您可以使用 `replication alter partitions sync` 设置设置等待。

可能的值: 0 – do not wait; 1 – only wait for own execution (default); 2 – wait for all.

突变

突变是允许更改或删除表中的行的ALTER查询变体。与标准相比 `UPDATE` 和 `DELETE` 用于点数据更改的查询, `mutations` 适用于更改表中大量行的繁重操作。支持的 `MergeTree` 表引擎系列, 包括具有复制支持的引擎。

现有表可以按原样进行突变 (无需转换), 但是在将第一次突变应用于表之后, 其元数据格式将与以前的服务器版本不兼容, 并且无法回退到以前的版本。

当前可用的命令:

```
ALTER TABLE [db.]table DELETE WHERE filter_expr
```

该 `filter_expr` 必须是类型 `UInt8`. 查询删除表中此表达式采用非零值的行。

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] WHERE filter_expr
```

该 `filter_expr` 必须是类型 `UInt8`. 此查询将指定列的值更新为行中相应表达式的值。`filter_expr` 取非零值。使用以下命令将值转换为列类型 `CAST` 接线员 不支持更新用于计算主键或分区键的列。

```
ALTER TABLE [db.]table MATERIALIZE INDEX name IN PARTITION partition_name
```

查询将重新生成二级索引 `name` 在分区中 `partition_name`.

一个查询可以包含多个用逗号分隔的命令。

For*`MergeTree`表的突变通过重写整个数据部分来执行。没有原子性-部分被取代为突变的部分, 只要他们准备好和 `SELECT` 在突变期间开始执行的查询将看到来自已经突变的部件的数据以及来自尚未突变的部件的数据。

突变完全按其创建顺序排序, 并以该顺序应用于每个部分。突变也使用插入进行部分排序-在提交突变之前插入到表中的数据将被突变, 之后插入的数据将不会被突变。请注意, 突变不会以任何方式阻止插入。

`Mutation`查询在添加mutation条目后立即返回 (如果将复制的表复制到ZooKeeper, 则将非复制的表复制到文件系统)。突变本身使用系统配置文件设置异步执行。要跟踪突变的进度, 您可以使用 `system.mutations` 桌子 即使重新启动 `ClickHouse`服务器, 成功提交的突变仍将继续执行。一旦提交, 没有办法回滚突变, 但如果突变由于某种原因被卡住, 可以使用 `KILL MUTATION` 查询。

已完成突变的条目不会立即删除 (保留条目的数量由 `finished_mutations_to_keep` 存储引擎参数)。旧的突变条目将被删除。

ALTER USER

更改`ClickHouse`用户帐户.

语法

```
ALTER USER [IF EXISTS] name [ON CLUSTER cluster_name]
[RENAME TO new_name]
[IDENTIFIED [WITH {PLAINTEXT_PASSWORD|SHA256_PASSWORD|DOUBLE_SHA1_PASSWORD}] BY
{'password'|'hash'}]
[[ADD|DROP] HOST {LOCAL | NAME 'name' | REGEXP 'name_regexp' | IP 'address' | LIKE 'pattern'} [,....] | ANY | NONE]
[DEFAULT ROLE role [,....] | ALL | ALL EXCEPT role [,....] ]
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | PROFILE
'profile_name'] [,...]
```

产品描述

使用 `ALTER USER` 你必须有 **ALTER USER** 特权

例

将授予的角色设置为默认值:

```
ALTER USER user DEFAULT ROLE role1, role2
```

如果以前未向用户授予角色，ClickHouse将引发异常。

将所有授予的角色设置为默认值:

```
ALTER USER user DEFAULT ROLE ALL
```

如果将来将某个角色授予某个用户，它将自动成为默认值。

将所有授予的角色设置为默认值，除非 `role1` 和 `role2`:

```
ALTER USER user DEFAULT ROLE ALL EXCEPT role1, role2
```

ALTER ROLE

更改角色。

语法

```
ALTER ROLE [IF EXISTS] name [ON CLUSTER cluster_name]
[RENAME TO new_name]
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | PROFILE
'profile_name'] [,...]
```

ALTER ROW POLICY

更改行策略。

语法

```
ALTER [ROW] POLICY [IF EXISTS] name [ON CLUSTER cluster_name] ON [database.]table
[RENAME TO new_name]
[AS {PERMISSIVE | RESTRICTIVE}]
[FOR SELECT]
[USING {condition | NONE}] [,...]
[TO {role [,...]} | ALL | ALL EXCEPT role [,...]]
```

ALTER QUOTA

更改配额。

语法

```
ALTER QUOTA [IF EXISTS] name [ON CLUSTER cluster_name]
[RENAME TO new_name]
[KEYED BY {'none' | 'user name' | 'ip address' | 'client key' | 'client key or user name' | 'client key or ip address'}]
[FOR [RANDOMIZED] INTERVAL number {SECOND | MINUTE | HOUR | DAY}
{MAX { {QUERIES | ERRORS | RESULT ROWS | RESULT BYTES | READ ROWS | READ BYTES | EXECUTION TIME} =
number } [,...]} |
```

```
NO LIMITS | TRACKING ONLY} [,...]]  
[TO {role [,...]} | ALL | ALL EXCEPT role [,...]}]
```

ALTER SETTINGS PROFILE

更改配额。

语法

```
ALTER SETTINGS PROFILE [IF EXISTS] name [ON CLUSTER cluster_name]  
[RENAME TO new_name]  
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | INHERIT  
'profile_name'] [,...]
```

系统查询

- RELOAD DICTIONARIES
- RELOAD DICTIONARY
- DROP DNS CACHE
- DROP MARK CACHE
- FLUSH LOGS
- RELOAD CONFIG
- SHUTDOWN
- KILL
- STOP DISTRIBUTED SENDS
- FLUSH DISTRIBUTED
- START DISTRIBUTED SENDS
- STOP MERGES
- START MERGES

RELOAD DICTIONARIES

重新加载之前已成功加载的所有字典。

默认情况下，字典是懒惰加载的（请参阅 [dictionaries_lazy_load](#)），所以不是在启动时自动加载，而是通过dictGet函数在第一次访问时初始化，或者从ENGINE=Dictionary的表中选择。该 `SYSTEM RELOAD DICTIONARIES` 查询重新加载这样的字典（加载）。

总是返回 `Ok.` 无论字典更新的结果如何。

重新加载字典Dictionary_name

完全重新加载字典 `dictionary_name`，与字典的状态无关（LOADED/NOT_LOADED/FAILED）。

总是返回 `Ok.` 无论更新字典的结果如何。

字典的状态可以通过查询 `system.dictionaries` 桌子

```
SELECT name, status FROM system.dictionaries;
```

DROP DNS CACHE

重置ClickHouse的内部DNS缓存。有时（对于旧的ClickHouse版本）在更改基础架构（更改另一个ClickHouse服务器或字典使用的服务器的IP地址）时需要使用此命令。

有关更方便（自动）缓存管理，请参阅[disable_internal_dns_cache](#)、[dns_cache_update_period](#)参数。

DROP MARK CACHE

重置标记缓存。用于开发ClickHouse和性能测试。

FLUSH LOGS

Flushes buffers of log messages to system tables (e.g. system.query_log). Allows you to not wait 7.5 seconds when debugging.

RELOAD CONFIG

重新加载ClickHouse配置。当配置存储在ZooKeeper中时使用。

SHUTDOWN

通常关闭ClickHouse（如 `service clickhouse-server stop / kill {$pid_clickhouse-server}`）

KILL

中止ClickHouse进程（如 `kill -9 {$pid_clickhouse-server}`）

管理分布式表

ClickHouse可以管理**分布**桌子。当用户将数据插入到这些表中时，ClickHouse首先创建应发送到群集节点的数据队列，然后异步发送它。您可以使用**STOP DISTRIBUTED SENDS**, **FLUSH DISTRIBUTED**, 和 **START DISTRIBUTED SENDS**查询。您也可以同步插入分布式数据与 `insert_distributed_sync` 设置。

STOP DISTRIBUTED SENDS

将数据插入分布式表时禁用后台数据分发。

```
SYSTEM STOP DISTRIBUTED SENDS [db.]<distributed_table_name>
```

FLUSH DISTRIBUTED

强制ClickHouse将数据同步发送到群集节点。如果任何节点不可用，ClickHouse将引发异常并停止查询执行。您可以重试查询，直到查询成功，这将在所有节点恢复联机时发生。

```
SYSTEM FLUSH DISTRIBUTED [db.]<distributed_table_name>
```

START DISTRIBUTED SENDS

将数据插入分布式表时启用后台数据分发。

```
SYSTEM START DISTRIBUTED SENDS [db.]<distributed_table_name>
```

STOP MERGES

提供停止MergeTree系列中表的后台合并的可能性：

```
SYSTEM STOP MERGES [[db.]]merge_tree_family_table_name]
```

注

`DETACH / ATTACH` 即使在之前所有MergeTree表的合并已停止的情况下，`table`也会为表启动后台合并。

START MERGES

为MergeTree系列中的表提供启动后台合并的可能性：

```
SYSTEM START MERGES [[db.]merge_tree_family_table_name]
```

显示查询

SHOW CREATE TABLE

```
SHOW CREATE [TEMPORARY] [TABLE|DICTIONARY] [db.]table [INTO OUTFILE filename] [FORMAT format]
```

返回单 `String`-类型 ‘statement’ column, which contains a single value – the `CREATE` 用于创建指定对象的查询。

SHOW DATABASES

```
SHOW DATABASES [INTO OUTFILE filename] [FORMAT format]
```

打印所有数据库的列表。

这个查询是相同的 `SELECT name FROM system.databases [INTO OUTFILE filename] [FORMAT format]`.

SHOW PROCESSLIST

```
SHOW PROCESSLIST [INTO OUTFILE filename] [FORMAT format]
```

输出的内容 `系统。流程` 表，包含目前正在处理的查询列表，除了 `SHOW PROCESSLIST` 查询。

该 `SELECT * FROM system.processes` 查询返回有关所有当前查询的数据。

提示（在控制台中执行）：

```
$ watch -n1 "clickhouse-client --query='SHOW PROCESSLIST'"
```

SHOW TABLES

显示表的列表。

```
SHOW [TEMPORARY] TABLES [{FROM | IN} <db>] [LIKE '<pattern>' | WHERE expr] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

如果 `FROM` 如果未指定子句，则查询返回当前数据库中的表列表。

你可以得到相同的结果 `SHOW TABLES` 通过以下方式进行查询：

```
SELECT name FROM system.tables WHERE database = <db> [AND name LIKE <pattern>] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

示例

下面的查询从表的列表中选择前两行 `system` 数据库，其名称包含 `co.`

```
SHOW TABLES FROM system LIKE '%co%' LIMIT 2
```

```
└── name  
    | aggregate_function_combinators |  
    | collations |
```

SHOW DICTIONARIES

显示列表 外部字典.

```
SHOW DICTIONARIES [FROM <db>] [LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

如果 FROM 如果未指定子句，则查询从当前数据库返回字典列表。

你可以得到相同的结果 SHOW DICTIONARIES 通过以下方式进行查询:

```
SELECT name FROM system.dictionaries WHERE database = <db> [AND name LIKE <pattern>] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

示例

下面的查询从表的列表中选择前两行 system 数据库，其名称包含 reg.

```
SHOW DICTIONARIES FROM db LIKE '%reg%' LIMIT 2
```

```
└── name  
    | regions |  
    | region_names |
```

SHOW GRANTS

显示用户的权限。

语法

```
SHOW GRANTS [FOR user]
```

如果未指定user，则查询返回当前用户的权限。

SHOW CREATE USER

显示了在使用的参数 用户创建.

SHOW CREATE USER 不输出用户密码。

语法

```
SHOW CREATE USER [name | CURRENT_USER]
```

SHOW CREATE ROLE

显示了在使用的参数 角色创建

语法

```
SHOW CREATE ROLE name
```

SHOW CREATE ROW POLICY

显示了在使用的参数 [创建行策略](#)

语法

```
SHOW CREATE [ROW] POLICY name ON [database.]table
```

SHOW CREATE QUOTA

显示了在使用的参数 [创建配额](#)

语法

```
SHOW CREATE QUOTA [name | CURRENT]
```

SHOW CREATE SETTINGS PROFILE

显示了在使用的参数 [设置配置文件创建](#)

语法

```
SHOW CREATE [SETTINGS] PROFILE name
```

GRANT

- Grants [privileges](#) to ClickHouse user accounts or roles.
- Assigns roles to user accounts or to the other roles.

To revoke privileges, use the [REVOKE](#) statement. Also you can list granted privileges with the [SHOW GRANTS](#) statement.

Granting Privilege Syntax

```
GRANT [ON CLUSTER cluster_name] privilege[(column_name [....])] [....] ON {db.table|db.*|*.*|table|*} TO {user | role | CURRENT_USER} [...] [WITH GRANT OPTION]
```

- `privilege` — Type of privilege.
- `role` — ClickHouse user role.
- `user` — ClickHouse user account.

The `WITH GRANT OPTION` clause grants `user` or `role` with permission to execute the `GRANT` query. Users can grant privileges of the same scope they have and less.

Assigning Role Syntax

```
GRANT [ON CLUSTER cluster_name] role [...] TO {user | another_role | CURRENT_USER} [...] [WITH ADMIN OPTION]
```

- `role` — ClickHouse user role.
- `user` — ClickHouse user account.

The `WITH ADMIN OPTION` clause grants **ADMIN OPTION** privilege to user or role.

Usage

To use `GRANT`, your account must have the `GRANT OPTION` privilege. You can grant privileges only inside the scope of your account privileges.

For example, administrator has granted privileges to the `john` account by the query:

```
GRANT SELECT(x,y) ON db.table TO john WITH GRANT OPTION
```

It means that `john` has the permission to execute:

- `SELECT x,y FROM db.table.`
- `SELECT x FROM db.table.`
- `SELECT y FROM db.table.`

`john` can't execute `SELECT z FROM db.table.` The `SELECT * FROM db.table` also is not available. Processing this query, ClickHouse doesn't return any data, even `x` and `y`. The only exception is if a table contains only `x` and `y` columns. In this case ClickHouse returns all the data.

Also `john` has the `GRANT OPTION` privilege, so it can grant other users with privileges of the same or smaller scope.

Specifying privileges you can use asterisk (*) instead of a table or a database name. For example, the `GRANT SELECT ON db.* TO john` query allows `john` to execute the `SELECT` query over all the tables in `db` database. Also, you can omit database name. In this case privileges are granted for current database. For example, `GRANT SELECT ON * TO john` grants the privilege on all the tables in the current database, `GRANT SELECT ON mytable TO john` grants the privilege on the `mytable` table in the current database.

Access to the `system` database is always allowed (since this database is used for processing queries).

You can grant multiple privileges to multiple accounts in one query. The query `GRANT SELECT, INSERT ON *.* TO john, robin` allows accounts `john` and `robin` to execute the `INSERT` and `SELECT` queries over all the tables in all the databases on the server.

Privileges

Privilege is a permission to execute specific kind of queries.

Privileges have a hierarchical structure. A set of permitted queries depends on the privilege scope.

Hierarchy of privileges:

- **SELECT**
- **INSERT**
- **ALTER**
 - **ALTER TABLE**
 - **ALTER UPDATE**
 - **ALTER DELETE**
 - **ALTER COLUMN**
 - **ALTER ADD COLUMN**
 - **ALTER DROP COLUMN**
 - **ALTER MODIFY COLUMN**
 - **ALTER COMMENT COLUMN**
 - **ALTER CLEAR COLUMN**
 - **ALTER RENAME COLUMN**
 - **ALTER INDEX**

- ALTER ORDER BY
- ALTER ADD INDEX
- ALTER DROP INDEX
- ALTER MATERIALIZE INDEX
- ALTER CLEAR INDEX
- ALTER CONSTRAINT
 - ALTER ADD CONSTRAINT
 - ALTER DROP CONSTRAINT
- ALTER TTL
- ALTER MATERIALIZE TTL
- ALTER SETTINGS
- ALTER MOVE PARTITION
- ALTER FETCH PARTITION
- ALTER FREEZE PARTITION
- ALTER VIEW
 - ALTER VIEW REFRESH
 - ALTER VIEW MODIFY QUERY
- CREATE
 - CREATE DATABASE
 - CREATE TABLE
 - CREATE VIEW
 - CREATE DICTIONARY
 - CREATE TEMPORARY TABLE
- DROP
 - DROP DATABASE
 - DROP TABLE
 - DROP VIEW
 - DROP DICTIONARY
- TRUNCATE
- OPTIMIZE
- SHOW
 - SHOW DATABASES
 - SHOW TABLES
 - SHOW COLUMNS
 - SHOW DICTIONARIES
- KILL QUERY
- ACCESS MANAGEMENT
 - CREATE USER
 - ALTER USER
 - DROP USER
 - CREATE ROLE
 - ALTER ROLE
 - DROP ROLE
 - CREATE ROW POLICY
 - ALTER ROW POLICY
 - DROP ROW POLICY
 - CREATE QUOTA
 - ALTER QUOTA
 - DROP QUOTA
 - CREATE SETTINGS PROFILE
 - ALTER SETTINGS PROFILE
 - DROP SETTINGS PROFILE
 - SHOW ACCESS
 - SHOW_USERS
 - SHOW_ROLES

- SHOW_RULES
- SHOW_ROW_POLICIES
- SHOW_QUOTAS
- SHOW_SETTINGS_PROFILES
- ROLE ADMIN
- SYSTEM
 - SYSTEM SHUTDOWN
 - SYSTEM DROP CACHE
 - SYSTEM DROP DNS CACHE
 - SYSTEM DROP MARK CACHE
 - SYSTEM DROP UNCOMPRESSED CACHE
 - SYSTEM RELOAD
 - SYSTEM RELOAD CONFIG
 - SYSTEM RELOAD DICTIONARY
 - SYSTEM RELOAD EMBEDDED DICTIONARIES
 - SYSTEM MERGES
 - SYSTEM TTL MERGES
 - SYSTEM FETCHES
 - SYSTEM MOVES
 - SYSTEM SENDS
 - SYSTEM DISTRIBUTED SENDS
 - SYSTEM REPLICATED SENDS
 - SYSTEM REPLICATION QUEUES
 - SYSTEM SYNC REPLICA
 - SYSTEM RESTART REPLICA
 - SYSTEM FLUSH
 - SYSTEM FLUSH DISTRIBUTED
 - SYSTEM FLUSH LOGS
- INTROSPECTION
 - addressToLine
 - addressToSymbol
 - demangle
- SOURCES
 - FILE
 - URL
 - REMOTE
 - YSQL
 - ODBC
 - JDBC
 - HDFS
 - S3
- dictGet

Examples of how this hierarchy is treated:

- The ALTER privilege includes all other ALTER* privileges.
- ALTER CONSTRAINT includes ALTER ADD CONSTRAINT and ALTER DROP CONSTRAINT privileges.

Privileges are applied at different levels. Knowing of a level suggests syntax available for privilege.

Levels (from lower to higher):

- COLUMN — Privilege can be granted for column, table, database, or globally.
- TABLE — Privilege can be granted for table, database, or globally.
- VIEW — Privilege can be granted for view, database, or globally.
- DICTIONARY — Privilege can be granted for dictionary, database, or globally.
- DATABASE — Privilege can be granted for database or globally.

- **DATABASE** — Privilege can be granted for database or globally.
- **GLOBAL** — Privilege can be granted only globally.
- **GROUP** — Groups privileges of different levels. When **GROUP**-level privilege is granted, only that privileges from the group are granted which correspond to the used syntax.

Examples of allowed syntax:

- GRANT SELECT(x) ON db.table TO user
- GRANT SELECT ON db.* TO user

Examples of disallowed syntax:

- GRANT CREATE USER(x) ON db.table TO user
- GRANT CREATE USER ON db.* TO user

The special privilege **ALL** grants all the privileges to a user account or a role.

By default, a user account or a role has no privileges.

If a user or a role has no privileges, it is displayed as **NONE** privilege.

Some queries by their implementation require a set of privileges. For example, to execute the **RENAME** query you need the following privileges: **SELECT**, **CREATE TABLE**, **INSERT** and **DROP TABLE**.

SELECT

Allows executing **SELECT** queries.

Privilege level: **COLUMN**.

Description

User granted with this privilege can execute **SELECT** queries over a specified list of columns in the specified table and database. If user includes other columns then specified a query returns no data.

Consider the following privilege:

```
GRANT SELECT(x,y) ON db.table TO john
```

This privilege allows **john** to execute any **SELECT** query that involves data from the **x** and/or **y** columns in **db.table**, for example, **SELECT x FROM db.table**. **john** can't execute **SELECT z FROM db.table**. The **SELECT * FROM db.table** also is not available. Processing this query, ClickHouse doesn't return any data, even **x** and **y**. The only exception is if a table contains only **x** and **y** columns, in this case ClickHouse returns all the data.

INSERT

Allows executing **INSERT** queries.

Privilege level: **COLUMN**.

Description

User granted with this privilege can execute **INSERT** queries over a specified list of columns in the specified table and database. If user includes other columns then specified a query doesn't insert any data.

Example

```
GRANT INSERT(x,y) ON db.table TO john
```

The granted privilege allows **john** to insert data to the **x** and/or **y** columns in **db.table**.

ALTER

Allows executing **ALTER** queries according to the following hierarchy of privileges:

- **ALTER**. Level: **COLUMN**.
 - **ALTER TABLE**. Level: **GROUP**
 - **ALTER UPDATE**. Level: **COLUMN**. Aliases: **UPDATE**
 - **ALTER DELETE**. Level: **COLUMN**. Aliases: **DELETE**
 - **ALTER COLUMN**. Level: **GROUP**
 - **ALTER ADD COLUMN**. Level: **COLUMN**. Aliases: **ADD COLUMN**
 - **ALTER DROP COLUMN**. Level: **COLUMN**. Aliases: **DROP COLUMN**
 - **ALTER MODIFY COLUMN**. Level: **COLUMN**. Aliases: **MODIFY COLUMN**
 - **ALTER COMMENT COLUMN**. Level: **COLUMN**. Aliases: **COMMENT COLUMN**
 - **ALTER CLEAR COLUMN**. Level: **COLUMN**. Aliases: **CLEAR COLUMN**
 - **ALTER RENAME COLUMN**. Level: **COLUMN**. Aliases: **RENAME COLUMN**
 - **ALTER INDEX**. Level: **GROUP**. Aliases: **INDEX**
 - **ALTER ORDER BY**. Level: **TABLE**. Aliases: **ALTER MODIFY ORDER BY**, **MODIFY ORDER BY**
 - **ALTER ADD INDEX**. Level: **TABLE**. Aliases: **ADD INDEX**
 - **ALTER DROP INDEX**. Level: **TABLE**. Aliases: **DROP INDEX**
 - **ALTER MATERIALIZE INDEX**. Level: **TABLE**. Aliases: **MATERIALIZE INDEX**
 - **ALTER CLEAR INDEX**. Level: **TABLE**. Aliases: **CLEAR INDEX**
 - **ALTER CONSTRAINT**. Level: **GROUP**. Aliases: **CONSTRAINT**
 - **ALTER ADD CONSTRAINT**. Level: **TABLE**. Aliases: **ADD CONSTRAINT**
 - **ALTER DROP CONSTRAINT**. Level: **TABLE**. Aliases: **DROP CONSTRAINT**
 - **ALTER TTL**. Level: **TABLE**. Aliases: **ALTER MODIFY TTL**, **MODIFY TTL**
 - **ALTER MATERIALIZE TTL**. Level: **TABLE**. Aliases: **MATERIALIZE TTL**
 - **ALTER SETTINGS**. Level: **TABLE**. Aliases: **ALTER SETTING**, **ALTER MODIFY SETTING**, **MODIFY SETTING**
 - **ALTER MOVE PARTITION**. Level: **TABLE**. Aliases: **ALTER MOVE PART**, **MOVE PARTITION**, **MOVE PART**
 - **ALTER FETCH PARTITION**. Level: **TABLE**. Aliases: **FETCH PARTITION**
 - **ALTER FREEZE PARTITION**. Level: **TABLE**. Aliases: **FREEZE PARTITION**
 - **ALTER VIEW** Level: **GROUP**
 - **ALTER VIEW REFRESH**. Level: **VIEW**. Aliases: **ALTER LIVE VIEW REFRESH**, **REFRESH VIEW**
 - **ALTER VIEW MODIFY QUERY**. Level: **VIEW**. Aliases: **ALTER TABLE MODIFY QUERY**

Examples of how this hierarchy is treated:

- The **ALTER** privilege includes all other **ALTER*** privileges.
- **ALTER CONSTRAINT** includes **ALTER ADD CONSTRAINT** and **ALTER DROP CONSTRAINT** privileges.

Notes

- The **MODIFY SETTING** privilege allows modifying table engine settings. It doesn't affect settings or server configuration parameters.
- The **ATTACH** operation needs the **CREATE** privilege.
- The **DETACH** operation needs the **DROP** privilege.
- To stop mutation by the **KILL MUTATION** query, you need to have a privilege to start this mutation. For example, if you want to stop the **ALTER UPDATE** query, you need the **ALTER UPDATE**, **ALTER TABLE**, or **ALTER** privilege.

CREATE

Allows executing **CREATE** and **ATTACH** DDL-queries according to the following hierarchy of privileges:

- **CREATE**. Level: **GROUP**
 - **CREATE DATABASE**. Level: **DATABASE**
 - **CREATE TABLE**. Level: **TABLE**
 - **CREATE VIEW**. Level: **VIEW**
 - **CREATE DICTIONARY**. Level: **DICTIONARY**

- CREATE TEMPORARY TABLE. Level: GLOBAL

Notes

- To delete the created table, a user needs DROP.

DROP

Allows executing DROP and DETACH queries according to the following hierarchy of privileges:

- DROP. Level:
 - DROP DATABASE. Level: DATABASE
 - DROP TABLE. Level: TABLE
 - DROP VIEW. Level: VIEW
 - DROP DICTIONARY. Level: DICTIONARY

TRUNCATE

Allows executing TRUNCATE queries.

Privilege level: TABLE.

OPTIMIZE

Allows executing OPTIMIZE TABLE queries.

Privilege level: TABLE.

SHOW

Allows executing SHOW, DESCRIBE, USE, and EXISTS queries according to the following hierarchy of privileges:

- SHOW. Level: GROUP
 - SHOW DATABASES. Level: DATABASE. Allows to execute SHOW DATABASES, SHOW CREATE DATABASE, USE <database> queries.
 - SHOW TABLES. Level: TABLE. Allows to execute SHOW TABLES, EXISTS <table>, CHECK <table> queries.
 - SHOW COLUMNS. Level: COLUMN. Allows to execute SHOW CREATE TABLE, DESCRIBE queries.
 - SHOW DICTIONARIES. Level: DICTIONARY. Allows to execute SHOW DICTIONARIES, SHOW CREATE DICTIONARY, EXISTS <dictionary> queries.

Notes

A user has the SHOW privilege if it has any other privilege concerning the specified table, dictionary or database.

KILL QUERY

Allows executing KILL queries according to the following hierarchy of privileges:

Privilege level: GLOBAL.

Notes

KILL QUERY privilege allows one user to kill queries of other users.

ACCESS MANAGEMENT

Allows a user to execute queries that manage users, roles and row policies.

- ACCESS MANAGEMENT. Level: GROUP
 - CREATE USER. Level: GLOBAL
 - ALTER USER. Level: GLOBAL
 - DROP USER. Level: GLOBAL
 - CREATE ROLE. Level: GLOBAL

- CREATE ROLE. Level: GLOBAL
- ALTER ROLE. Level: GLOBAL
- DROP ROLE. Level: GLOBAL
- ROLE ADMIN. Level: GLOBAL
- CREATE ROW POLICY. Level: GLOBAL. Aliases: CREATE POLICY
- ALTER ROW POLICY. Level: GLOBAL. Aliases: ALTER POLICY
- DROP ROW POLICY. Level: GLOBAL. Aliases: DROP POLICY
- CREATE QUOTA. Level: GLOBAL
- ALTER QUOTA. Level: GLOBAL
- DROP QUOTA. Level: GLOBAL
- CREATE SETTINGS PROFILE. Level: GLOBAL. Aliases: CREATE PROFILE
- ALTER SETTINGS PROFILE. Level: GLOBAL. Aliases: ALTER PROFILE
- DROP SETTINGS PROFILE. Level: GLOBAL. Aliases: DROP PROFILE
- SHOW ACCESS. Level: GROUP
 - SHOW_USERS. Level: GLOBAL. Aliases: SHOW CREATE USER
 - SHOW_ROLES. Level: GLOBAL. Aliases: SHOW CREATE ROLE
 - SHOW_ROW_POLICIES. Level: GLOBAL. Aliases: SHOW POLICIES, SHOW CREATE ROW POLICY, SHOW CREATE POLICY
 - SHOW_QUOTAS. Level: GLOBAL. Aliases: SHOW CREATE QUOTA
 - SHOW_SETTINGS_PROFILES. Level: GLOBAL. Aliases: SHOW PROFILES, SHOW CREATE SETTINGS PROFILE, SHOW CREATE PROFILE

The ROLE ADMIN privilege allows a user to assign and revoke any roles including those which are not assigned to the user with the admin option.

SYSTEM

Allows a user to execute **SYSTEM** queries according to the following hierarchy of privileges.

- SYSTEM. Level: GROUP
 - SYSTEM SHUTDOWN. Level: GLOBAL. Aliases: SYSTEM KILL, SHUTDOWN
 - SYSTEM DROP CACHE. Aliases: DROP CACHE
 - SYSTEM DROP DNS CACHE. Level: GLOBAL. Aliases: SYSTEM DROP DNS, DROP DNS CACHE, DROP DNS
 - SYSTEM DROP MARK CACHE. Level: GLOBAL. Aliases: SYSTEM DROP MARK, DROP MARK CACHE, DROP MARKS
 - SYSTEM DROP UNCOMPRESSED CACHE. Level: GLOBAL. Aliases: SYSTEM DROP UNCOMPRESSED, DROP UNCOMPRESSED CACHE, DROP UNCOMPRESSED
 - SYSTEM RELOAD. Level: GROUP
 - SYSTEM RELOAD CONFIG. Level: GLOBAL. Aliases: RELOAD CONFIG
 - SYSTEM RELOAD DICTIONARY. Level: GLOBAL. Aliases: SYSTEM RELOAD DICTIONARIES, RELOAD DICTIONARY, RELOAD DICTIONARIES
 - SYSTEM RELOAD EMBEDDED DICTIONARIES. Level: GLOBAL. Aliases: RELOAD EMBEDDED DICTIONARIES
 - SYSTEM MERGES. Level: TABLE. Aliases: SYSTEM STOP MERGES, SYSTEM START MERGES, STOP MERGES, START MERGES
 - SYSTEM TTL MERGES. Level: TABLE. Aliases: SYSTEM STOP TTL MERGES, SYSTEM START TTL MERGES, STOP TTL MERGES, START TTL MERGES
 - SYSTEM FETCHES. Level: TABLE. Aliases: SYSTEM STOP FETCHES, SYSTEM START FETCHES, STOP FETCHES, START FETCHES
 - SYSTEM MOVES. Level: TABLE. Aliases: SYSTEM STOP MOVES, SYSTEM START MOVES, STOP MOVES, START MOVES
 - SYSTEM SENDS. Level: GROUP. Aliases: SYSTEM STOP SENDS, SYSTEM START SENDS, STOP SENDS, START SENDS
 - SYSTEM DISTRIBUTED SENDS. Level: TABLE. Aliases: SYSTEM STOP DISTRIBUTED SENDS, SYSTEM START DISTRIBUTED SENDS, STOP DISTRIBUTED SENDS, START DISTRIBUTED SENDS
 - SYSTEM DUPLICATED SENDS. Level: TABLE. Aliases: SYSTEM STOP DUPLICATED SENDS, SYSTEM START DUPLICATED SENDS

- SYSTEM REPLICATED SENDS. Level: TABLE. Aliases: SYSTEM STOP REPLICATED SENDS, SYSTEM START REPLICATED SENDS, STOP REPLICATED SENDS, START REPLICATED SENDS
- SYSTEM REPLICATION QUEUES. Level: TABLE. Aliases: SYSTEM STOP REPLICATION QUEUES, SYSTEM START REPLICATION QUEUES, STOP REPLICATION QUEUES, START REPLICATION QUEUES
- SYSTEM SYNC REPLICA. Level: TABLE. Aliases: SYNC REPLICA
- SYSTEM RESTART REPLICA. Level: TABLE. Aliases: RESTART REPLICA
- SYSTEM FLUSH. Level: GROUP
 - SYSTEM FLUSH DISTRIBUTED. Level: TABLE. Aliases: FLUSH DISTRIBUTED
 - SYSTEM FLUSH LOGS. Level: GLOBAL. Aliases: FLUSH LOGS

The SYSTEM RELOAD EMBEDDED DICTIONARIES privilege implicitly granted by the SYSTEM RELOAD DICTIONARY ON *.* privilege.

INTROSPECTION

Allows using **introspection** functions.

- INTROSPECTION. Level: GROUP. Aliases: INTROSPECTION FUNCTIONS
 - addressToLine. Level: GLOBAL
 - addressToSymbol. Level: GLOBAL
 - demangle. Level: GLOBAL

SOURCES

Allows using external data sources. Applies to **table engines** and **table functions**.

- SOURCES. Level: GROUP
 - FILE. Level: GLOBAL
 - URL. Level: GLOBAL
 - REMOTE. Level: GLOBAL
 - YSQL. Level: GLOBAL
 - ODBC. Level: GLOBAL
 - JDBC. Level: GLOBAL
 - HDFS. Level: GLOBAL
 - S3. Level: GLOBAL

The SOURCES privilege enables use of all the sources. Also you can grant a privilege for each source individually. To use sources, you need additional privileges.

Examples:

- To create a table with the **MySQL table engine**, you need CREATE TABLE (ON db.table_name) and **MySQL** privileges.
- To use the **mysql table function**, you need CREATE TEMPORARY TABLE and **MySQL** privileges.

dictGet

- dictGet. Aliases: dictHas, dictGetHierarchy, dictIsIn

Allows a user to execute **dictGet**, **dictHas**, **dictGetHierarchy**, **dictIsIn** functions.

Privilege level: **DICTIONARY**.

Examples

- GRANT dictGet ON mydb.mydictionary TO john
- GRANT dictGet ON mydictionary TO john

ALL

Grants all the privileges on regulated entity to a user account or a role.

NONE

Doesn't grant any privileges.

ADMIN OPTION

The `ADMIN OPTION` privilege allows a user to grant their role to another user.

REVOKE

Revokes privileges from users or roles.

Syntax

Revoking privileges from users

```
REVOKE [ON CLUSTER cluster_name] privilege[(column_name [...])] [...] ON {db.table|db.*|*.*|table|*} FROM {user | CURRENT_USER} [...] | ALL | ALL EXCEPT {user | CURRENT_USER} [...]
```

Revoking roles from users

```
REVOKE [ON CLUSTER cluster_name] [ADMIN OPTION FOR] role [...] FROM {user | role | CURRENT_USER} [...] | ALL | ALL EXCEPT {user_name | role_name | CURRENT_USER} [...]
```

Description

To revoke some privilege you can use a privilege of a wider scope than you plan to revoke. For example, if a user has the `SELECT (x,y)` privilege, administrator can execute `REVOKE SELECT(x,y) ...`, or `REVOKE SELECT * ...`, or even `REVOKE ALL PRIVILEGES ...` query to revoke this privilege.

Partial Revokes

You can revoke a part of a privilege. For example, if a user has the `SELECT *.*` privilege you can revoke from it a privilege to read data from some table or a database.

Examples

Grant the `john` user account with a privilege to select from all the databases, excepting the `accounts` one:

```
GRANT SELECT ON *.* TO john;
REVOKE SELECT ON accounts.* FROM john;
```

Grant the `mira` user account with a privilege to select from all the columns of the `accounts.staff` table, excepting the `wage` one.

```
GRANT SELECT ON accounts.staff TO mira;
REVOKE SELECT(wage) ON accounts.staff FROM mira;
```

杂项查询

ATTACH

这个查询是完全一样的 `CREATE`，但是

- 而不是这个词 `CREATE` 它使用这个词 `ATTACH`.
- 查询不会在磁盘上创建数据，但假定数据已经在适当的位置，只是将有关表的信息添加到服务器。执行附加查询后，服务器将知道表的存在。

如果表之前已分离 (`DETACH`)，意味着其结构是已知的，可以使用速记而不限定该结构。

```
ATTACH TABLE [IF NOT EXISTS] [db.]name [ON CLUSTER cluster]
```

启动服务器时使用此查询。服务器将表元数据作为文件存储 `ATTACH` 查询，它只是在启动时运行（除了在服务器上显式创建的系统表）。

CHECK TABLE

检查表中的数据是否已损坏。

```
CHECK TABLE [db.]name
```

该 `CHECK TABLE` 查询将实际文件大小与存储在服务器上的预期值进行比较。如果文件大小与存储的值不匹配，则表示数据已损坏。例如，这可能是由查询执行期间的系统崩溃引起的。

查询响应包含 `result` 具有单行的列。该行的值为

布尔值 类型：

- 0-表中的数据已损坏。
- 1-数据保持完整性。

该 `CHECK TABLE` 查询支持下表引擎：

- 日志
- `TinyLog`
- `StripeLog`
- 梅树家族

使用另一个表引擎对表执行会导致异常。

从发动机 `*Log` 家庭不提供故障自动数据恢复。使用 `CHECK TABLE` 查询以及时跟踪数据丢失。

为 `MergeTree` 家庭发动机，`CHECK TABLE` 查询显示本地服务器上表的每个单独数据部分的检查状态。

如果数据已损坏

如果表已损坏，则可以将未损坏的数据复制到另一个表。要做到这一点：

1. 创建具有与损坏的表相同结构的新表。要执行此操作，请执行查询 `CREATE TABLE <new_table_name> AS <damaged_table_name>`.
2. 设置 `max_threads` 值为 1 以便在单个线程中处理下一个查询。要执行此操作，请运行查询 `SET max_threads = 1`.
3. 执行查询 `INSERT INTO <new_table_name> SELECT * FROM <damaged_table_name>`. 此请求将未损坏的数据从损坏的表复制到另一个表。只有损坏部分之前的数据才会被复制。
4. 重新启动 `clickhouse-client` 要重置 `max_threads` 价值。

DESCRIBE TABLE

```
DESC|DESCRIBE TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

返回以下内容 `String` 类型列：

- `name` — Column name.

- `type` — Column type.
- `default_type` — Clause that is used in **默认表达式** (`DEFAULT`, `MATERIALIZED` 或 `ALIAS`). 如果未指定默认表达式，则Column包含一个空字符串。
- `default_expression` — Value specified in the `DEFAULT` 条款
- `comment_expression` — Comment text.

嵌套的数据结构输出 “expanded” 格式。每列分别显示，名称后面有一个点。

DETACH

删除有关 ‘name’ 表从服务器。服务器停止了解表的存在。

```
DETACH TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

这不会删除表的数据或元数据。在下一次服务器启动时，服务器将读取元数据并再次查找有关表的信息。
同样，一个 “detached” 表可以使用重新连接 `ATTACH` 查询（系统表除外，它们没有为它们存储元数据）。

没有 `DETACH DATABASE` 查询。

DROP

此查询有两种类型: `DROP DATABASE` 和 `DROP TABLE`.

```
DROP DATABASE [IF EXISTS] db [ON CLUSTER cluster]
```

删除内部的所有表 ‘db’ 数据库，然后删除 ‘db’ 数据库本身。

如果 `IF EXISTS` 如果数据库不存在，则不会返回错误。

```
DROP [TEMPORARY] TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

删除表。

如果 `IF EXISTS` 如果表不存在或数据库不存在，则不会返回错误。

```
DROP DICTIONARY [IF EXISTS] [db.]name
```

删除字典。

如果 `IF EXISTS` 如果表不存在或数据库不存在，则不会返回错误。

DROP USER

删除用户。

语法

```
DROP USER [IF EXISTS] name [, ...] [ON CLUSTER cluster_name]
```

DROP ROLE

删除角色。

已删除的角色将从授予该角色的所有实体撤销。

语法

```
DROP ROLE [IF EXISTS] name [, ...] [ON CLUSTER cluster_name]
```

DROP ROW POLICY

删除行策略。

已删除行策略将从分配该策略的所有实体撤销。

语法

```
DROP [ROW] POLICY [IF EXISTS] name [,....] ON [database.]table [,....] [ON CLUSTER cluster_name]
```

DROP QUOTA

删除配额。

已删除的配额将从分配配额的所有实体撤销。

语法

```
DROP QUOTA [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

DROP SETTINGS PROFILE

删除配额。

已删除的配额将从分配配额的所有实体撤销。

语法

```
DROP [SETTINGS] PROFILE [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

EXISTS

```
EXISTS [TEMPORARY] [TABLE|DICTIONARY] [db.]name [INTO OUTFILE filename] [FORMAT format]
```

返回单 `UInt8`-type column，其中包含单个值 `0` 如果表或数据库不存在，或 `1` 如果该表存在于指定的数据库中。

KILL QUERY

```
KILL QUERY [ON CLUSTER cluster]
WHERE <where expression to SELECT FROM system.processes query>
[SYNC|ASYNC|TEST]
[FORMAT format]
```

尝试强制终止当前正在运行的查询。

要终止的查询是从系统中选择的。使用在定义的标准进程表 `WHERE` 《公约》条款 `KILL` 查询。

例：

```
-- Forcibly terminates all queries with the specified query_id:
KILL QUERY WHERE query_id='2-857d-4a57-9ee0-327da5d60a90'

-- Synchronously terminates all queries run by 'username':
KILL QUERY WHERE user='username' SYNC
```

默认情况下，使用异步版本的查询 (**ASYNC**)，不等待确认查询已停止。

同步版本 (**SYNC**) 等待所有查询停止，并在停止时显示有关每个进程的信息。

响应包含 `kill_status` 列，它可以采用以下值：

1. ‘finished’ – The query was terminated successfully.
2. ‘waiting’ – Waiting for the query to end after sending it a signal to terminate.
3. The other values explain why the query can't be stopped.

测试查询 (**TEST**) 仅检查用户的权限并显示要停止的查询列表。

KILL MUTATION

```
KILL MUTATION [ON CLUSTER cluster]
WHERE <where expression to SELECT FROM system.mutations query>
[TEST]
[FORMAT format]
```

尝试取消和删除 **突变** 当前正在执行。要取消的突变选自 `system.mutations` 表使用由指定的过滤器 `WHERE` 《公约》条款 `KILL` 查询。

测试查询 (**TEST**) 仅检查用户的权限并显示要停止的查询列表。

例：

```
-- Cancel and remove all mutations of the single table:
KILL MUTATION WHERE database = 'default' AND table = 'table'

-- Cancel the specific mutation:
KILL MUTATION WHERE database = 'default' AND table = 'table' AND mutation_id = 'mutation_3.txt'
```

The query is useful when a mutation is stuck and cannot finish (e.g. if some function in the mutation query throws an exception when applied to the data contained in the table).

已经由突变所做的更改不会回滚。

OPTIMIZE

```
OPTIMIZE TABLE [db.]name [ON CLUSTER cluster] [PARTITION partition | PARTITION ID 'partition_id'] [FINAL]
[DEDUPLICATE]
```

此查询尝试使用来自表引擎的表初始化表的数据部分的非计划合并 **MergeTree** 家人

该 `OPTIMIZE` 查询也支持 **MaterializedView** 和 **缓冲区** 引擎 不支持其他表引擎。

当 `OPTIMIZE` 与使用 **ReplicatedMergeTree** 表引擎的家族，ClickHouse创建合并任务，并等待在所有节点上执行（如果 `replication_alter_partitions_sync` 设置已启用）。

- 如果 `OPTIMIZE` 出于任何原因不执行合并，它不通知客户端。要启用通知，请使用 `optimize_throw_if_noop` 设置。
- 如果您指定 `PARTITION`，仅优化指定的分区。如何设置分区表达式.
- 如果您指定 `FINAL`，即使所有数据已经在一部分中，也会执行优化。
- 如果您指定 `DEDUPLICATE`，然后完全相同的行将被重复数据删除（所有列进行比较），这仅适用于**MergeTree**引擎。

警告

`OPTIMIZE` 无法修复 “Too many parts” 错误

RENAME

重命名一个或多个表。

```
RENAME TABLE [db11.]name11 TO [db12.]name12, [db21.]name21 TO [db22.]name22, ... [ON CLUSTER cluster]
```

所有表都在全局锁定下重命名。重命名表是一个轻型操作。如果您在TO之后指定了另一个数据库，则表将被移动到此数据库。但是，包含数据库的目录必须位于同一文件系统中（否则，将返回错误）。

SET

```
SET param = value
```

分配 `value` 到 `param` 设置 对于当前会话。你不能改变 [服务器设置](#) 这边

您还可以在单个查询中设置指定设置配置文件中的所有值。

```
SET profile = 'profile-name-from-the-settings-file'
```

有关详细信息，请参阅 [设置](#).

SET ROLE

激活当前用户的角色。

语法

```
SET ROLE {DEFAULT | NONE | role [,...] | ALL | ALL EXCEPT role [,...]}
```

SET DEFAULT ROLE

将默认角色设置为用户。

默认角色在用户登录时自动激活。您只能将以前授予的角色设置为默认值。如果未向用户授予角色，ClickHouse将引发异常。

语法

```
SET DEFAULT ROLE {NONE | role [,...] | ALL | ALL EXCEPT role [,...]} TO {user|CURRENT_USER} [,...]
```

例

为用户设置多个默认角色:

```
SET DEFAULT ROLE role1, role2, ... TO user
```

将所有授予的角色设置为用户的默认值:

```
SET DEFAULT ROLE ALL TO user
```

从用户清除默认角色:

```
SET DEFAULT ROLE NONE TO user
```

将所有授予的角色设置为默认角色，其中一些角色除外：

```
SET DEFAULT ROLE ALL EXCEPT role1, role2 TO user
```

TRUNCATE

```
TRUNCATE TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

从表中删除所有数据。当条款 **IF EXISTS** 如果该表不存在，则查询返回错误。

该 **TRUNCATE** 查询不支持 **查看**, **文件**, **URL** 和 **Null** 表引擎。

USE

```
USE db
```

用于设置会话的当前数据库。

当前数据库用于搜索表，如果数据库没有在查询中明确定义与表名之前的点。

使用HTTP协议时无法进行此查询，因为没有会话的概念。

CREATE DATABASE

该查询用于根据指定名称创建数据库。

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

数据库其实只是用于存放表的一个目录。

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

CREATE TABLE

对于 **CREATE TABLE**，存在以下几种方式。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = engine
```

在指定的'db'数据库中创建一个名为'name'的表，如果查询中没有包含'db'，则默认使用当前选择的数据库作为'db'。后面的是包含在括号中的表结构以及表引擎的声明。

其中表结构声明是一个包含一组列描述声明的组合。如果表引擎是支持索引的，那么可以在表引擎的参数中对其进行说明。

在最简单的情况下，列描述是指 **名称** **类型** 这样的子句。例如：**RegionID UInt32**。

但是也可以为列另外定义默认值表达式（见后文）。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name AS [db2.]name2 [ENGINE = engine]
```

创建一个与 **db2.name2** 具有相同结构的表，同时你可以对其指定不同的表引擎声明。如果没有表引擎声明，则创建的表将与 **db2.name2** 使用相同的表引擎。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name ENGINE = engine AS SELECT ...
```

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

以上所有情况，如果指定了 `IF NOT EXISTS`，那么在该表已经存在的情况下，查询不会返回任何错误。在这种情况下，查询几乎不会做任何事情。

在 `ENGINE` 子句后还可能存在一些其他的子句，更详细的信息可以参考 [表引擎](#) 中关于建表的描述。

默认值

在列描述中你可以通过以下方式之一为列指定默认表达式：`DEFAULT expr`，`MATERIALIZED expr`，`ALIAS expr`。

示例：`URLDomain String DEFAULT domain(URL)`。

如果在列描述中未定义任何默认表达式，那么系统将会根据类型设置对应的默认值，如：数值类型为零、字符串类型为空字符串、数组类型为空数组、日期类型为'0000-00-00'以及时间类型为'0000-00-00 00:00:00'。不支持使用NULL作为普通类型的默认值。

如果定义了默认表达式，则可以不定义列的类型。如果没有明确的定义类的类型，则使用默认表达式的类型。例如：`EventDate DEFAULT toDate(EventTime)` - 最终'EventDate'将使用'Date'作为类型。

如果同时指定了默认表达式与列的类型，则将使用类型转换函数将默认表达式转换为指定的类型。例如：`Hits UInt32 DEFAULT 0` 与 `Hits UInt32 DEFAULT toUInt32(0)` 意思相同。

默认表达式可以包含常量或表的任意其他列。当创建或更改表结构时，系统将会运行检查，确保不会包含循环依赖。对于 `INSERT`，它仅检查表达式是否是可以解析的 - 它们可以从中计算出所有需要的列的默认值。

`DEFAULT expr`

普通的默认值，如果 `INSERT` 中不包含指定的列，那么将通过表达式计算它的默认值并填充它。

`MATERIALIZED expr`

物化表达式，被该表达式指定的列不能包含在 `INSERT` 的列表中，因为它总是被计算出来的。

对于 `INSERT` 而言，不需要考虑这些列。

另外，在 `SELECT` 查询中如果包含星号，此列不会被用来替换星号，这是因为考虑到数据转储，在使用 `SELECT *` 查询出的结果总能够被 '`INSERT`' 回表。

`ALIAS expr`

别名。这样的列不会存储在表中。

它的值不能够通过 `INSERT` 写入，同时使用 `SELECT` 查询星号时，这些列也不会被用来替换星号。

但是它们可以显示的用于 `SELECT` 中，在这种情况下，在查询分析中别名将被替换。

当使用 `ALTER` 查询对添加新的列时，不同于为所有旧数据添加这个列，对于需要在旧数据中查询新列，只会在查询时动态计算这个新列的值。但是如果新列的默认表示中依赖其他列的值进行计算，那么同样会加载这些依赖的列的数据。

如果你向表中添加一个新列，并在之后的一段时间后修改它的默认表达式，则旧数据中的值将会被改变。请注意，在运行后台合并时，缺少的列的值将被计算后写入到合并后的数据部分中。

不能够为 `nested` 类型的列设置默认值。

制约因素

随着列描述约束可以定义：

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1],
    ...
)
```

```
CONSTRAINT constraint_name_1 CHECK boolean_expr_1,  
...  
) ENGINE = engine
```

`boolean_expr_1` 可以通过任何布尔表达式。如果为表定义了约束，则将为表中的每一行检查它们中的每一行 `INSERT` query. If any constraint is not satisfied — server will raise an exception with constraint name and checking expression.

添加大量的约束会对big的性能产生负面影响 `INSERT` 查询。

TTL表达式

定义值的存储时间。只能为MergeTree系列表指定。有关详细说明，请参阅 [列和表的TTL](#).

列压缩编解ecs

默认情况下，ClickHouse应用以下定义的压缩方法 [服务器设置](#)，列。您还可以定义在每个单独的列的压缩方法 `CREATE TABLE` 查询。

```
CREATE TABLE codec_example  
(  
    dt Date CODEC(ZSTD),  
    ts DateTime CODEC(LZ4HC),  
    float_value Float32 CODEC(NONE),  
    double_value Float64 CODEC(LZ4HC(9))  
    value Float32 CODEC(Delta, ZSTD)  
)  
ENGINE = <Engine>  
...
```

如果指定了编解ec，则默认编解码器不适用。编解码器可以组合在一个流水线中，例如，CODEC(Delta, ZSTD). 要为您的项目选择最佳的编解码器组合，请通过类似于Altinity中描述的基准测试 [新编码提高ClickHouse效率](#) 文章.

警告

您无法使用外部实用程序解压缩ClickHouse数据库文件，如 lz4. 相反，使用特殊的环板**compressor** 实用程序。

下表引擎支持压缩：

- [MergeTree](#) 家庭
- [日志](#) 家庭
- [设置](#)
- [加入我们](#)

ClickHouse支持通用编解码器和专用编解ecs。

专业编解ecs

这些编解码器旨在通过使用数据的特定功能使压缩更有效。其中一些编解码器不压缩数据本身。相反，他们准备的数据用于共同目的的编解ec，其压缩它比没有这种准备更好。

专业编解ecs:

- `Delta(delta_bytes)` — Compression approach in which raw values are replaced by the difference of two neighboring values, except for the first value that stays unchanged. Up to `delta_bytes` 用于存储增量值，所以 `delta_bytes` 是原始值的最大大小。可能 `delta_bytes` 值:1,2,4,8. 默认值 `delta_bytes` 是 `sizeof(type)` 如果等于 1, 2, 4或8。在所有其他情况下，它是1。
- `DoubleDelta` — Calculates delta of deltas and writes it in compact binary form. Optimal compression

- **DoubleDelta** — Calculates delta of deltas and writes it in compact binary form. Optimal compression rates are achieved for monotonic sequences with a constant stride, such as time series data. Can be used with any fixed-width type. Implements the algorithm used in Gorilla TSDB, extending it to support 64-bit types. Uses 1 extra bit for 32-byte deltas: 5-bit prefixes instead of 4-bit prefixes. For additional information, see Compressing Time Stamps in Gorilla : 一个快速、可扩展的内存时间序列数据库.
- **Gorilla** — Calculates XOR between current and previous value and writes it in compact binary form. Efficient when storing a series of floating point values that change slowly, because the best compression rate is achieved when neighboring values are binary equal. Implements the algorithm used in Gorilla TSDB, extending it to support 64-bit types. For additional information, see Compressing Values in Gorilla : 一个快速、可扩展的内存时间序列数据库.
- **T64** — Compression approach that crops unused high bits of values in integer data types (including `Enum`, `Date` 和 `DateTime`). 在算法的每个步骤中，编解码器采用64个值块，将它们放入 64×64 位矩阵中，对其进行转置，裁剪未使用的位并将其余部分作为序列返回。未使用的位是使用压缩的整个数据部分的最大值和最小值之间没有区别的位。

`DoubleDelta` 和 `Gorilla` 编解码器在Gorilla TSDB中用作其压缩算法的组件。大猩猩的方法是有效的情况下，当有缓慢变化的值与他们的时间戳序列。时间戳是由有效地压缩 `DoubleDelta` 编解ec，和值有效地由压缩 `Gorilla` 编解ec 例如，要获取有效存储的表，可以在以下配置中创建它：

```
CREATE TABLE codec_example
(
    timestamp DateTime CODEC(DoubleDelta),
    slow_values Float32 CODEC(Gorilla)
)
ENGINE = MergeTree()
```

通用编解ecs

编解ecs:

- **NONE** — No compression.
- **LZ4** — Lossless 数据压缩算法 默认情况下使用。应用LZ4快速压缩。
- **LZ4HC[(level)]** — LZ4 HC (high compression) algorithm with configurable level. Default level: 9. Setting level ≤ 0 应用默认级别。可能的水平：[1, 12]。推荐级别范围：[4, 9]。
- **ZSTD[(level)]** — ZSTD压缩算法 可配置 level. 可能的水平：[1, 22]。默认值：1。

高压缩级别对于非对称场景非常有用，例如压缩一次，重复解压缩。更高的级别意味着更好的压缩和更高的CPU使用率。

临时表

ClickHouse支持临时表，其具有以下特征：

- 当会话结束时，临时表将随会话一起消失，这包含链接中断。
- 临时表仅能够使用Memory表引擎。
- 无法为临时表指定数据库。它是在数据库之外创建的。
- 如果临时表与另一个表名称相同，那么当在查询时没有显示的指定db的情况下，将优先使用临时表。
- 对于分布式处理，查询中使用的临时表将被传递到远程服务器。

可以使用下面的语法创建一个临时表：

```
CREATE TEMPORARY TABLE [IF NOT EXISTS] table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
)
```

大多数情况下，临时表不是手动创建的，只有在分布式查询处理中使用 (GLOBAL) IN 时为外部数据创建。更多信息，可以参考

相关章节。

分布式DDL查询（ON CLUSTER 子句）

对于 CREATE, DROP, ALTER, 以及 RENAME 查询，系统支持其运行在整个集群上。

例如，以下查询将在 cluster 集群的所有节点上创建名为 all_hits 的 Distributed 表：

```
CREATE TABLE IF NOT EXISTS all_hits ON CLUSTER cluster (p Date, i Int32) ENGINE = Distributed(cluster, default, hits)
```

为了能够正确的运行这种查询，每台主机必须具有相同的cluster声明（为了简化配置的同步，你可以使用zookeeper的方式进行配置）。同时这些主机还必须链接到zookeeper服务器。

这个查询将最终在集群的每台主机上运行，即使一些主机当前处于不可用状态。同时它还保证了所有的查询在单台主机中的执行顺序。

CREATE VIEW

```
CREATE [MATERIALIZED] VIEW [IF NOT EXISTS] [db.]table_name [TO[db.]name] [ENGINE = engine] [POPULATE] AS  
SELECT ...
```

创建一个视图。它存在两种可选择的类型：普通视图与物化视图。

普通视图不存储任何数据，只是执行从另一个表中的读取。换句话说，普通视图只是保存了视图的查询，当从视图中查询时，此查询被作为子查询用于替换FROM子句。

举个例子，假设你已经创建了一个视图：

```
CREATE VIEW view AS SELECT ...
```

还有一个查询：

```
SELECT a, b, c FROM view
```

这个查询完全等价于：

```
SELECT a, b, c FROM (SELECT ...)
```

物化视图存储的数据是由相应的SELECT查询转换得来的。

在创建物化视图时，你还必须指定表的引擎 - 将会使用这个表引擎存储数据。

目前物化视图的工作原理：当将数据写入到物化视图中SELECT子句所指定的表时，插入的数据会通过SELECT子句查询进行转换并将最终结果插入到视图中。

如果创建物化视图时指定了POPULATE子句，则在创建时将该表的数据插入到物化视图中。就像使用CREATE TABLE ... AS SELECT ...一样。否则，物化视图只会包含在物化视图创建后的新写入的数据。我们不推荐使用POPULATE，因为在视图创建期间写入的数据将不会写入其中。

当一个SELECT子句包含DISTINCT, GROUP BY, ORDER BY, LIMIT时，请注意，这些仅会在插入数据时在每个单独的数据块上执行。例如，如果你在其中包含了GROUP BY，则只会在查询期间进行聚合，但聚合范围仅限于单个批的写入数据。数据不会进一步被聚合。但是当你使用一些其他数据聚合引擎时这是例外的，如：SummingMergeTree。

目前对物化视图执行ALTER是不支持的，因此这可能是不方便的。如果物化视图是使用的TO [db.]name的方式进行构建的，你可以使用DETACH语句先将视图剥离，然后使用ALTER运行在目标表上，然后使用ATTACH将之前剥离的表重新加载进来。

视图看起来和普通的表相同。例如，你可以通过SHOW TABLES查看到它们。

没有单独的删除视图的语法。如果要删除视图，请使用 `DROP TABLE`。

CREATE DICTIONARY

```
CREATE DICTIONARY [IF NOT EXISTS] [db.]dictionary_name [ON CLUSTER cluster]
(
    key1 type1 [DEFAULT|EXPRESSION expr1] [HIERARCHICAL|INJECTIVE|IS_OBJECT_ID],
    key2 type2 [DEFAULT|EXPRESSION expr2] [HIERARCHICAL|INJECTIVE|IS_OBJECT_ID],
    attr1 type2 [DEFAULT|EXPRESSION expr3],
    attr2 type2 [DEFAULT|EXPRESSION expr4]
)
PRIMARY KEY key1, key2
SOURCE(SOURCE_NAME([param1 value1 ... paramN valueN]))
LAYOUT(LAYOUT_NAME([param_name param_value]))
LIFETIME([MIN val1] MAX val2)
```

INSERT

`INSERT`查询主要用于向系统中添加数据。

查询的基本格式：

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

您可以在查询中指定插入的列的列表，如：`[(c1, c2, c3)]`。对于存在于表结构中但不存在于插入列表中的列，它们将会按照如下方式填充数据：

- 如果存在 `DEFAULT` 表达式，根据 `DEFAULT` 表达式计算被填充的值。
- 如果没有定义 `DEFAULT` 表达式，则填充零或空字符串。

如果 `strict_insert_defaults=1`，你必须在查询中列出所有没有定义 `DEFAULT` 表达式的列。

数据可以以ClickHouse支持的任何 [输入输出格式](#) 传递给`INSERT`。格式的名称必须显示的指定在查询中：

```
INSERT INTO [db.]table [(c1, c2, c3)] FORMAT format_name data_set
```

例如，下面的查询所使用的输入格式就与上面`INSERT ... VALUES`的中使用的输入格式相同：

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

ClickHouse会清除数据前所有的空白字符与一行摘要信息（如果需要的话）。所以在进行查询时，我们建议您将数据放入到输入输出格式名称后的新的一行中去（如果数据是以空白字符开始的，这将非常重要）。

示例：

```
INSERT INTO t FORMAT TabSeparated
11 Hello, world!
22 Qwerty
```

在使用命令行客户端或HTTP客户端时，你可以将具体的查询语句与数据分开发送。更多具体信息，请参考[«客户端»部分](#)。

使用 `SELECT` 的结果写入

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

写入与SELECT的列的对应关系是使用位置来进行对应的，尽管它们在SELECT表达式与INSERT中的名称可能是不同的。如果需要，会对它们执行对应的类型转换。

除了VALUES格式之外，其他格式中的数据都不允许出现诸如now()， $1 + 2$ 等表达式。VALUES格式允许您有限度的使用这些表达式，但是不建议您这么做，因为执行这些表达式总是低效的。

系统不支持的其他用于修改数据的查询：UPDATE，DELETE，REPLACE，MERGE，UPSERT，INSERT UPDATE。
但是，您可以使用 ALTER TABLE ... DROP PARTITION查询来删除一些旧的数据。

性能的注意事项

在进行INSERT时将会对写入的数据进行一些处理，按照主键排序，按照月份对数据进行分区等。所以如果在您的写入数据中包含多个月份的混合数据时，将会显著的降低INSERT的性能。为了避免这种情况：

- 数据总是以尽量大的batch进行写入，如每次写入100,000行。
- 数据在写入ClickHouse前预先的对数据进行分组。

在以下的情况下，性能不会下降：

- 数据总是被实时的写入。
- 写入的数据已经按照时间排序。

语法

系统中有两种类型的解析器：完整SQL解析器（递归下降解析器）和数据格式解析器（快速流解析器）。

在所有情况下，除了 INSERT 查询时，只使用完整的SQL解析器。

该 INSERT 查询使用两个解析器：

```
INSERT INTO t VALUES (1, 'Hello, world'), (2, 'abc'), (3, 'def')
```

该 INSERT INTO t VALUES 片段由完整的解析器解析，并且数据 (1, 'Hello, world'), (2, 'abc'), (3, 'def') 由快速流解析器解析。您也可以通过使用 `input_format_values_interpret_expressions` 设置。当 `input_format_values_interpret_expressions = 1`，ClickHouse首先尝试使用fast stream解析器解析值。如果失败，ClickHouse将尝试对数据使用完整的解析器，将其视为SQL 表达式。

数据可以有任何格式。当接收到查询时，服务器计算不超过 `max_query_size` RAM中请求的字节（默认为1MB），其余的是流解析。

它允许避免与大的问题 INSERT 查询。

使用时 Values 格式为 INSERT 查询，它可能看起来数据被解析相同的表达式 SELECT 查询，但事实并非如此。该 Values 格式更为有限。

本文的其余部分将介绍完整的解析器。有关格式解析器的详细信息，请参阅 [格式](#) 科。

空间

语法结构之间可能有任意数量的空格符号（包括查询的开始和结束）。空格符号包括空格、制表符、换行符、CR和换页符。

评论

ClickHouse支持SQL风格和C风格的注释。

SQL风格的注释以下开头 -- 并继续到线的末尾，一个空格后 -- 可以省略。

C型是从 /* 到 */并且可以是多行，也不需要空格。

关键词

当关键字对应于以下关键字时，不区分大小写：

- SQL标准。例如, `SELECT`, `select` 和 `SeLeCt` 都是有效的。
- 在一些流行的DBMS (MySQL或Postgres) 中实现。例如, `DateTime` 是一样的 `datetime`.

数据类型名称是否区分大小写可以在 `system.data_type_families` 桌子

与标准SQL相比，所有其他关键字（包括函数名称）都是 区分大小写.

不保留关键字；它们仅在相应的上下文中被视为保留关键字。如果您使用 **标识符** 使用与关键字相同的名称，将它们括在双引号或反引号中。例如，查询 `SELECT "FROM" FROM table_name` 是有效的，如果表 `table_name` 具有名称的列 `"FROM"`.

标识符

标识符是:

- 集群、数据库、表、分区和列名称。
- 功能。
- 数据类型。
- 表达式别名.

标识符可以是引号或非引号。后者是优选的。

非引号标识符必须与正则表达式匹配 `^[a-zA-Z_][0-9a-zA-Z_]*$` 并且不能等于 **关键词**. 例: `x, _1, X_y_Z123_`.

如果要使用与关键字相同的标识符，或者要在标识符中使用其他符号，请使用双引号或反引号对其进行引用，例如, `"id"`, ``id``.

文字数

有数字，字符串，复合和 `NULL` 文字。

数字

数值文字尝试进行分析:

- 首先，作为一个64位有符号的数字，使用 `strtoull` 功能。
- 如果不成功，作为64位无符号数，使用 `strtoll` 功能。
- 如果不成功，作为一个浮点数使用 `strtod` 功能。
- 否则，将返回错误。

文本值具有该值适合的最小类型。

例如，`1`被解析为 `UInt8`，但`256`被解析为 `UInt16`. 有关详细信息，请参阅 **数据类型**.

例: `1, 18446744073709551615, 0xDEADBEEF, 01, 0.1, 1e100, -1e-100, inf, nan.`

字符串

仅支持单引号中的字符串文字。封闭的字符可以反斜杠转义。以下转义序列具有相应的特殊值: `\b`, `\f`, `\r`, `\n`, `\t`, `\0`, `\a`, `\v`, `\xHH`. 在所有其他情况下，转义序列的格式为 `\c`，哪里 `c` 是任何字符，被转换为 `c`. 这意味着你可以使用序列 `\`` 和 `\```. 该值将具有 **字符串** 类型。

在字符串文字中，你至少需要转义 `'` 和 `\``. 单引号可以用单引号，文字转义 `'\t's'` 和 `'\t"''s'` 是平等的。

化合物

数组使用方括号构造 `[1, 2, 3]`. Nuples用圆括号构造 `(1, 'Hello, world!', 2)`.

从技术上讲，这些不是文字，而是分别具有数组创建运算符和元组创建运算符的表达式。

数组必须至少包含一个项目，元组必须至少包含两个项目。

有一个单独的情况下，当元组出现在 `IN a`条款 `SELECT` 查询。查询结果可以包含元组，但元组不能保存到数据库（除了具有以下内容的表 **记忆** 发动机）。

NULL

指示该值丢失。

为了存储 `NULL` 在表字段中，它必须是 可为空 类型。

根据数据格式（输入或输出），`NULL` 可能有不同的表示。有关详细信息，请参阅以下文档 [数据格式](#)。

处理有许多细微差别 `NULL`。例如，如果比较操作的至少一个参数是 `NULL`，此操作的结果也是 `NULL`。对于乘法，加法和其他操作也是如此。有关详细信息，请阅读每个操作的文档。

在查询中，您可以检查 `NULL` 使用 `IS NULL` 和 `IS NOT NULL` 运算符及相关功能 `isNull` 和 `isNotNull`。

功能

函数调用像一个标识符一样写入，并在圆括号中包含一个参数列表（可能是空的）。与标准SQL相比，括号是必需的，即使是空的参数列表。示例：`now()`。

有常规函数和聚合函数（请参阅部分“Aggregate functions”）。某些聚合函数可以包含括号中的两个参数列表。示例：`quantile (0.9) (x)`。这些聚合函数被调用“parametric”函数，并在第一个列表中的参数被调用“parameters”。不带参数的聚合函数的语法与常规函数的语法相同。

运营商

在查询解析过程中，运算符会转换为相应的函数，同时考虑它们的优先级和关联性。

例如，表达式 `1 + 2 * 3 + 4` 转化为 `plus(plus(1, multiply(2, 3)), 4)`。

数据类型和数据库表引擎

数据类型和表引擎 `CREATE` 查询的编写方式与标识符或函数相同。换句话说，它们可能包含也可能不包含括号中的参数列表。有关详细信息，请参阅部分“Data types,” “Table engines,” 和“CREATE”。

表达式别名

别名是查询中表达式的用户定义名称。

`expr AS alias`

- `AS` — The keyword for defining aliases. You can define the alias for a table name or a column name in a `SELECT` 子句不使用 `AS` 关键字。

For example, `SELECT table_name_alias.column_name FROM table_name table_name_alias`.

In the [CAST](sql_reference/functions/type_conversion_functions.md#type_conversion_function-cast) function, the `AS` keyword has another meaning. See the description of the function.

- `expr` — Any expression supported by ClickHouse.

For example, `SELECT column_name * 2 AS double FROM some_table`.

- `alias` — Name for `expr`. 别名应符合 [标识符](#) 语法

For example, `SELECT "table t".column_name FROM table_name AS "table t"`.

使用注意事项

别名对于查询或子查询是全局的，您可以在查询的任何部分中为任何表达式定义别名。例如，`SELECT (1 AS n) + 2, n`。

别名在子查询和子查询之间不可见。例如，在执行查询时 `SELECT (SELECT sum(b.a) + num FROM b) - a.a AS num FROM a` ClickHouse生成异常 `Unknown identifier: num`。

如果为结果列定义了别名 `SELECT` 子查询的子句，这些列在外部查询中可见。例如，`SELECT n + m FROM (SELECT 1 AS n, 2 AS m).`

小心使用与列或表名相同的别名。让我们考虑以下示例：

```
CREATE TABLE t
(
    a Int,
    b Int
)
ENGINE = TinyLog()
```

```
SELECT
    argMax(a, b),
    sum(b) AS b
FROM t
```

Received exception from server (version 18.14.17):

Code: 184. DB::Exception: Received from localhost:9000, 127.0.0.1. DB::Exception: Aggregate function sum(b) is found inside another aggregate function in query.

在这个例子中，我们声明表 `t` 带柱 `b`。然后，在选择数据时，我们定义了 `sum(b) AS b` 别名 由于别名是全局的，ClickHouse 替换了文字 `b` 在表达式中 `argMax(a, b)` 用表达式 `sum(b)`。这种替换导致异常。

星号

在一个 `SELECT` 查询中，星号可以替换表达式。有关详细信息，请参阅部分“`SELECT`”。

表达式

表达式是函数、标识符、文字、运算符的应用程序、括号中的表达式、子查询或星号。它还可以包含别名。

表达式列表是一个或多个用逗号分隔的表达式。

函数和运算符，反过来，可以有表达式作为参数。

聚合函数

聚合函数在 **正常** 方式如预期的数据库专家。

ClickHouse还支持：

- **参数聚合函数**，它接受除列之外的其他参数。
- **组合器**，这改变了聚合函数的行为。

空处理

在聚合过程中，所有 `NULLs` 被跳过。

例：

考虑这个表：

x	y
1	2
2	NULL
3	2

3	3
3	NULL

比方说，你需要在总的值 `y` 列:

```
SELECT sum(y) FROM t_null_big
```

sum(y)
7

该 `sum` 函数解释 `NULL` 作为 `0`. 特别是，这意味着，如果函数接收输入的选择，其中所有的值 `NULL`，那么结果将是 `0`，不 `NULL`.

现在你可以使用 `groupArray` 函数从创建一个数组 `y` 列:

```
SELECT groupArray(y) FROM t_null_big
```

groupArray(y)
[2,2,3]

`groupArray` 不包括 `NULL` 在生成的数组中。

聚合函数引用 计数

计数行数或非空值。

ClickHouse 支持以下语法 `count`:

- `count(expr)` 或 `COUNT(DISTINCT expr)`.
- `count()` 或 `COUNT(*)`. 该 `count()` 语法是 ClickHouse 特定的。

参数

该功能可以采取:

- 零参数。
- 一 表达式.

返回值

- 如果没有参数调用函数，它会计算行数。
- 如果 表达式 被传递，则该函数计数此表达式返回的次数非null。如果表达式返回 可为空 键入值，然后结果 `count` 保持不 `Nullable`. 如果返回表达式，则该函数返回 `0` `NULL` 对于所有的行。

在这两种情况下，返回值的类型为 `UInt64`.

详细信息

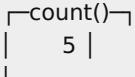
ClickHouse 支持 `COUNT(DISTINCT ...)` 语法 这种结构的行为取决于 `count_distinctImplementation` 设置。它定义了其中的 `uniq*` 函数用于执行操作。默认值为 `uniqExact` 功能。

该 `SELECT count() FROM table` 查询未被优化，因为表中的条目数没有单独存储。它从表中选择一个小列并计算其中的值数。

例

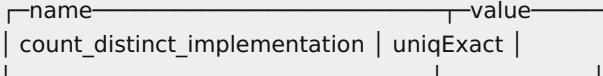
示例1:

```
SELECT count() FROM t
```

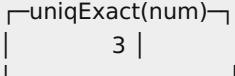


示例2:

```
SELECT name, value FROM system.settings WHERE name = 'count_distinctImplementation'
```



```
SELECT count(DISTINCT num) FROM t
```



这个例子表明 `count(DISTINCT num)` 由执行 `uniqExact` 根据功能 `count_distinctImplementation` 设定值。

任何(x)

选择第一个遇到的值。

查询可以以任何顺序执行，甚至每次都以不同的顺序执行，因此此函数的结果是不确定的。

要获得确定的结果，您可以使用 '`min`' 或 '`max`' 功能，而不是 '`any`'.

在某些情况下，可以依靠执行的顺序。这适用于 `SELECT` 来自使用 `ORDER BY` 的子查询的情况。

当一个 `SELECT` 查询具有 `GROUP BY` 子句或至少一个聚合函数，ClickHouse（相对于MySQL）要求在所有表达式 `SELECT`, `HAVING`, 和 `ORDER BY` 子句可以从键或聚合函数计算。换句话说，从表中选择的每个列必须在键或聚合函数内使用。要获得像MySQL这样的行为，您可以将其他列放在 `any` 聚合函数。

anyHeavy(x)

使用选择一个频繁出现的值 **重打者** 算法。如果某个值在查询的每个执行线程中出现的情况超过一半，则返回此值。通常情况下，结果是不确定的。

```
anyHeavy(column)
```

参数

- `column` – The column name.

示例

就拿 **时间** 数据集，并选择在任何频繁出现的值 **AirlineID** 列。

```
SELECT anyHeavy(AirlineID) AS res  
FROM ontime
```

```
└── res  
| 19690 |
```

anyLast(x)

选择遇到的最后一个值。

其结果是一样不确定的 **any** 功能。

集团比特

按位应用 **AND** 对于一系列的数字。

```
groupBitAnd(expr)
```

参数

expr – An expression that results in **UInt*** 类型。

返回值

的值 **UInt*** 类型。

示例

测试数据：

```
binary    decimal  
00101100 = 44  
00011100 = 28  
00001101 = 13  
01010101 = 85
```

查询：

```
SELECT groupBitAnd(num) FROM t
```

哪里 **num** 是包含测试数据的列。

结果：

```
binary    decimal  
00000100 = 4
```

groupBitOr

按位应用 **OR** 对于一系列的数字。

```
groupBitOr(expr)
```

参数

`expr` – An expression that results in `UInt*` 类型。

返回值

的值 `UInt*` 类型。

示例

测试数据:

```
binary    decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

查询:

```
SELECT groupBitOr(num) FROM t
```

哪里 `num` 是包含测试数据的列。

结果:

```
binary    decimal
01111101 = 125
```

groupBitXor

按位应用 `XOR` 对于一系列的数字。

```
groupBitXor(expr)
```

参数

`expr` – An expression that results in `UInt*` 类型。

返回值

的值 `UInt*` 类型。

示例

测试数据:

```
binary    decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

查询:

```
SELECT groupBitXor(num) FROM t
```

哪里 `num` 是包含测试数据的列。

结果:

```
binary    decimal  
01101000 = 104
```

groupBitmap

从无符号整数列的位图或聚合计算，返回 UInt64类型的基数，如果添加后缀状态，则返回 [位图对象](#).

```
groupBitmap(expr)
```

参数

`expr` – An expression that results in `UInt*` 类型。

返回值

的值 `UInt64` 类型。

示例

测试数据:

```
UserID  
1  
1  
2  
3
```

查询:

```
SELECT groupBitmap(UserID) as num FROM t
```

结果:

```
num  
3
```

min(x)

计算最小值。

max(x)

计算最大值。

argMin(arg,val)

计算 ‘arg’ 最小值的值 ‘val’ 价值。如果有几个不同的值 ‘arg’ 对于最小值 ‘val’，遇到的第一个值是输出。

user	salary
director	5000
manager	3000
worker	1000

```
SELECT argMin(user, salary) FROM salary
```

argMin(user, salary)	
worker	

argMax(arg,val)

计算 ‘arg’ 最大值 ‘val’ 价值。如果有几个不同的值 ‘arg’ 对于最大值 ‘val’，遇到的第一个值是输出。

sum(x)

计算总和。

只适用于数字。

sumWithOverflow(x)

使用与输入参数相同的数据类型计算数字的总和。如果总和超过此数据类型的最大值，则函数返回错误。

只适用于数字。

sumMap(key,value),sumMap(Tuple(key,value))

总计 ‘value’ 数组根据在指定的键 ‘key’ 阵列。

传递键和值数组的元组与传递两个键和值数组是同义的。

元素的数量 ‘key’ 和 ‘value’ 总计的每一行必须相同。

Returns a tuple of two arrays: keys in sorted order, and values summed for the corresponding keys.

示例：

```
CREATE TABLE sum_map(
    date Date,
    timeslot DateTime,
    statusMap Nested(
        status UInt16,
        requests UInt64
    ),
    statusMapTuple Tuple(Array(Int32), Array(Int32))
) ENGINE = Log;
INSERT INTO sum_map VALUES
    ('2000-01-01', '2000-01-01 00:00:00', [1, 2, 3], [10, 10, 10], ([1, 2, 3], [10, 10, 10])),
    ('2000-01-01', '2000-01-01 00:00:00', [3, 4, 5], [10, 10, 10], ([3, 4, 5], [10, 10, 10])),
    ('2000-01-01', '2000-01-01 00:01:00', [4, 5, 6], [10, 10, 10], ([4, 5, 6], [10, 10, 10])),
    ('2000-01-01', '2000-01-01 00:01:00', [6, 7, 8], [10, 10, 10], ([6, 7, 8], [10, 10, 10]));

SELECT
    timeslot,
    sumMap(statusMap.status, statusMap.requests),
    sumMap(statusMapTuple)
FROM sum_map
```

GROUP BY timeslot

```
timeslot—sumMap(statusMap.status, statusMap.requests)—sumMap(statusMapTuple)
| 2000-01-01 00:00:00 | ([1,2,3,4,5],[10,10,20,10,10]) | ([1,2,3,4,5],[10,10,20,10,10]) |
| 2000-01-01 00:01:00 | ([4,5,6,7,8],[10,10,20,10,10]) | ([4,5,6,7,8],[10,10,20,10,10]) |
```

skewPop

计算歪斜的序列。

`skewPop(expr)`

参数

`expr` — 表达式 返回一个数字。

返回值

The skewness of the given distribution. Type — **Float64**

示例

```
SELECT skewPop(value) FROM series_with_value_column
```

skewSamp

计算 样品偏度 的序列。

它表示随机变量的偏度的无偏估计，如果传递的值形成其样本。

`skewSamp(expr)`

参数

expr — 表达式 返回一个数字。

返回值

The skewness of the given distribution. Type — `Float64`. 如果 $n \leq 1$ (n 是样本的大小)，则该函数返回 `nan`.

示例

```
SELECT skewSamp(value) FROM series_with_value_column
```

kurtPop

计算 峰度 的序列。

kurtPop(expr)

参数

`expr` — 表达式 返回一个数字。

返回值

The kurtosis of the given distribution. Type — `Float64`

示例

```
SELECT kurtPop(value) FROM series_with_value_column
```

kurtSamp

计算 峰度样本 的序列。

它表示随机变量峰度的无偏估计，如果传递的值形成其样本。

```
kurtSamp(expr)
```

参数

`expr` — 表达式 返回一个数字。

返回值

The kurtosis of the given distribution. Type — `Float64`. 如果 `n <= 1` (`n` 是样本的大小)，则该函数返回 `nan`.

示例

```
SELECT kurtSamp(value) FROM series_with_value_column
```

timeSeriesGroupSum(uid,timestamp,value)

`timeSeriesGroupSum` 可以聚合不同的时间序列，即采样时间戳不对齐。

它将在两个采样时间戳之间使用线性插值，然后将时间序列和在一起。

- `uid` 是时间序列唯一id, `UInt64`.
- `timestamp` 是`Int64`型，以支持毫秒或微秒。
- `value` 是指标。

函数返回元组数组 `(timestamp, aggregated_value)` 对。

在使用此功能之前，请确保 `timestamp` 按升序排列

示例：

uid	timestamp	value
1	2	0.2
1	7	0.7
1	12	1.2
1	17	1.7
1	25	2.5
2	3	0.6
2	8	1.6
2	12	2.4
2	18	3.6
2	24	4.8

```

CREATE TABLE time_series(
    uid      UInt64,
    timestamp Int64,
    value    Float64
) ENGINE = Memory;
INSERT INTO time_series VALUES
(1,2,0.2),(1,7,0.7),(1,12,1.2),(1,17,1.7),(1,25,2.5),
(2,3,0.6),(2,8,1.6),(2,12,2.4),(2,18,3.6),(2,24,4.8);

SELECT timeSeriesGroupSum(uid, timestamp, value)
FROM (
    SELECT * FROM time_series order by timestamp ASC
);

```

其结果将是：

```
[(2,0.2),(3,0.9),(7,2.1),(8,2.4),(12,3.6),(17,5.1),(18,5.4),(24,7.2),(25,2.5)]
```

timeSeriesGroupRateSum(uid,ts,val)

同样 `timeSeriesGroupSum`, `timeSeriesGroupRateSum` 计算时间序列的速率，然后将速率总和在一起。
此外，使用此函数之前，时间戳应该是上升顺序。

应用此功能从数据 `timeSeriesGroupSum` 例如，您将得到以下结果：

```
[(2,0),(3,0.1),(7,0.3),(8,0.3),(12,0.3),(17,0.3),(18,0.3),(24,0.3),(25,0.1)]
```

avg(x)

计算平均值。

只适用于数字。

结果总是 `Float64`。

平均加权

计算 **加权算术平均值**.

语法

```
avgWeighted(x, weight)
```

参数

- `x` — Values. 整数 或 浮点.
- `weight` — Weights of the values. 整数 或 浮点.

类型 `x` 和 `weight` 一定是一样的

返回值

- 加权平均值。
- `Nan`. 如果所有的权重都等于0。

类型: `Float64`.

示例

查询:

```
SELECT avgWeighted(x, w)
FROM values('x Int8, w Int8', (4, 1), (1, 0), (10, 2))
```

结果:

```
└─avgWeighted(x, weight)─
   └─ 8 ─
```

uniq

计算参数的不同值的近似数量。

```
uniq(x[, ...])
```

参数

该函数采用可变数量的参数。参数可以是 `Tuple`, `Array`, `Date`, `DateTime`, `String`，或数字类型。

返回值

- A `UInt64`-键入号码。

实施细节

功能:

- 计算聚合中所有参数的哈希值，然后在计算中使用它。
- 使用自适应采样算法。对于计算状态，该函数使用最多 65536 个元素哈希值的样本。

This algorithm is very accurate and very efficient on the CPU. When the query contains several of these functions, using `uniq` is almost as fast as using other aggregate functions.

- 确定性地提供结果（它不依赖于查询处理顺序）。

我们建议在几乎所有情况下使用此功能。

另请参阅

- [uniqCombined](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)

uniqCombined

计算不同参数值的近似数量。

```
uniqCombined(HLL_precision)(x[, ...])
```

该 `uniqCombined` 函数是计算不同数值数量的不错选择。

参数

该函数采用可变数量的参数。参数可以是 `Tuple`, `Array`, `Date`, `DateTime`, `String`，或数字类型。

该函数不用于数组里的参数。参数可以是 `Tuple`, `Array`, `Date`, `DateTime`, `String`，或数字类型。

`HLL_precision` 是以2为底的单元格数的对数 **HyperLogLog**。可选，您可以将该函数用作 `uniqCombined(x[, ...])`。默认值 `HLL_precision` 是17，这是有效的96KiB的空间（ 2^{17} 个单元，每个6比特）。

返回值

- 一个数字 **UInt64**-键入号码。

实施细节

功能：

- 计算散列（64位散列 `String` 否则32位）对于聚合中的所有参数，然后在计算中使用它。
- 使用三种算法的组合：数组、哈希表和**HyperLogLog**与**error**错表。

For a small number of distinct elements, an array is used. When the set size is larger, a hash table is used. For a larger number of elements, HyperLogLog is used, which will occupy a fixed amount of memory.

- 确定性地提供结果（它不依赖于查询处理顺序）。

注

因为它使用32位散列非-`String` 类型，结果将有非常高的误差基数显着大于 `UINT_MAX`（错误将在几百亿不同值之后迅速提高），因此在这种情况下，您应该使用 **uniqCombined64**

相比于 `uniq` 功能，该 `uniqCombined`：

- 消耗少几倍的内存。
- 计算精度高出几倍。
- 通常具有略低的性能。在某些情况下，`uniqCombined` 可以表现得比 `uniq`，例如，使用通过网络传输大量聚合状态的分布式查询。

另请参阅

- [uniq](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)

uniqCombined64

和 `uniqCombined`，但对所有数据类型使用64位哈希。

uniqHLL12

计算不同参数值的近似数量，使用 **HyperLogLog** 算法。

`uniqHLL12(x[, ...])`

参数

该函数采用可变数量的参数。参数可以是 `Tuple`, `Array`, `Date`, `DateTime`, `String`，或数字类型。

返回值

- A **UInt64**-键入号码。

实施细节

功能:

- 计算聚合中所有参数的哈希值，然后在计算中使用它。
- 使用HyperLogLog算法来近似不同参数值的数量。

212 5-bit cells are used. The size of the state is slightly more than 2.5 KB. The result is not very accurate (up to ~10% error) for small data sets (<10K elements). However, the result is fairly accurate for high-cardinality data sets (10K-100M), with a maximum error of ~1.6%. Starting from 100M, the estimation error increases, and the function will return very inaccurate results for data sets with extremely high cardinality (1B+ elements).

- 提供确定结果（它不依赖于查询处理顺序）。

我们不建议使用此功能。在大多数情况下，使用 `uniq` 或 `uniqCombined` 功能。

另请参阅

- `uniq`
- `uniqCombined`
- `uniqExact`

uniqExact

计算不同参数值的准确数目。

```
uniqExact(x[, ...])
```

使用 `uniqExact` 功能，如果你绝对需要一个确切的结果。否则使用 `uniq` 功能。

该 `uniqExact` 功能使用更多的内存比 `uniq`，因为状态的大小随着不同值的数量的增加而无界增长。

参数

该函数采用可变数量的参数。参数可以是 `Tuple`, `Array`, `Date`, `DateTime`, `String`，或数字类型。

另请参阅

- `uniq`
- `uniqCombined`
- `uniqHLL12`

群交(`x`),群交(`max_size`)(`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。

第二个版本（与 `max_size` 参数）将结果数组的大小限制为 `max_size` 元素。

例如，`groupArray(1)(x)` 相当于 `[any(x)]`.

在某些情况下，您仍然可以依靠执行的顺序。这适用于以下情况 `SELECT` 来自使用 `ORDER BY`。

groupArrayInsertAt

在指定位置向数组中插入一个值。

语法

```
groupArrayInsertAt(default_x, size)(x, pos);
```

如果在一个查询中将多个值插入到同一位置，则该函数的行为方式如下：

- 如果在单个线程中执行查询，则使用第一个插入的值。
- 如果在多个线程中执行查询，则结果值是未确定的插入值之一。

参数

- `x` — Value to be inserted. 表达式 导致的一个 支持的数据类型.
- `pos` — Position at which the specified element `x` 将被插入。数组中的索引编号从零开始。 UInt32.
- `default_x`— Default value for substituting in empty positions. Optional parameter. 表达式 导致为配置的数据类型 `x` 参数。如果 `default_x` 未定义，则 默认值 被使用。
- `size`— Length of the resulting array. Optional parameter. When using this parameter, the default value `default_x` 必须指定。 UInt32.

返回值

- 具有插入值的数组。

类型: 阵列.

示例

查询:

```
SELECT groupArrayInsertAt(toString(number), number * 2) FROM numbers(5);
```

结果:

```
└─groupArrayInsertAt(toString(number), multiply(number, 2))─  
| ['0','1','2','3','4'] |
```

查询:

```
SELECT groupArrayInsertAt('')(toString(number), number * 2) FROM numbers(5);
```

结果:

```
└─groupArrayInsertAt('')(toString(number), multiply(number, 2))─  
| ['0','1','2','3','4'] |
```

查询:

```
SELECT groupArrayInsertAt('', 5)(toString(number), number * 2) FROM numbers(5);
```

结果:

```
└─groupArrayInsertAt('', 5)(toString(number), multiply(number, 2))─  
| ['0','1','2'] |
```

元件的多线程插入到一个位置。

查询:

```
SELECT groupArrayInsertAt(number, 0) FROM numbers_mt(10) SETTINGS max_block_size = 1;
```

作为这个查询的结果，你会得到随机整数 [0,9] 范围。例如:

```
└─groupArrayInsertAt(number, 0)─  
| [7] |
```

groupArrayMovingSum

计算输入值的移动和。

```
groupArrayMovingSum(numbers_for_summing)  
groupArrayMovingSum(window_size)(numbers_for_summing)
```

该函数可以将窗口大小作为参数。如果未指定，则该函数的窗口大小等于列中的行数。

参数

- `numbers_for_summing` — 表达式 生成数值数据类型值。
- `window_size` — Size of the calculation window.

返回值

- 与输入数据大小和类型相同的数组。

示例

样品表:

```
CREATE TABLE t  
(  
    `int` UInt8,  
    `float` Float32,  
    `dec` Decimal32(2)  
)  
ENGINE = TinyLog
```

```
└─int──float──dec─  
| 1 | 1.1 | 1.10 |  
| 2 | 2.2 | 2.20 |  
| 4 | 4.4 | 4.40 |  
| 7 | 7.77 | 7.77 |
```

查询:

```
SELECT  
    groupArrayMovingSum(int) AS I,  
    groupArrayMovingSum(float) AS F,  
    groupArrayMovingSum(dec) AS D  
FROM t
```

F	D
[1,3,7,14] [1.1,3.3000002,7.7000003,15.47]	[1.10,3.30,7.70,15.47]

```
SELECT
    groupArrayMovingSum(2)(int) AS I,
    groupArrayMovingSum(2)(float) AS F,
    groupArrayMovingSum(2)(dec) AS D
FROM t
```

F	D
[1,3,6,11] [1.1,3.3000002,6.6000004,12.17]	[1.10,3.30,6.60,12.17]

groupArrayMovingAvg

计算输入值的移动平均值。

```
groupArrayMovingAvg(numbers_for_summing)
groupArrayMovingAvg(window_size)(numbers_for_summing)
```

该函数可以将窗口大小作为参数。如果未指定，则该函数的窗口大小等于列中的行数。

参数

- numbers_for_summing — 表达式 生成数值数据类型值。
- window_size — Size of the calculation window.

返回值

- 与输入数据大小和类型相同的数组。

该函数使用 四舍五入到零。它截断结果数据类型的小数位数。

示例

样品表 `b`:

```
CREATE TABLE t
(
    `int` UInt8,
    `float` Float32,
    `dec` Decimal32(2)
)
ENGINE = TinyLog
```

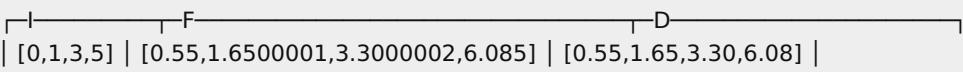
int	float	dec
1	1.1	1.10
2	2.2	2.20
4	4.4	4.40
7	7.77	7.77

查询:

```
SELECT
    groupArrayMovingAvg(int) AS I,
    groupArrayMovingAvg(float) AS F,
    groupArrayMovingAvg(dec) AS D
FROM t
```



```
SELECT
    groupArrayMovingAvg(2)(int) AS I,
    groupArrayMovingAvg(2)(float) AS F,
    groupArrayMovingAvg(2)(dec) AS D
FROM t
```



禄, 赖麓ta脣麓,) : 艇,, 拢脢, group媒group) galaxy s8碌脢脢) 禄煤) 酶脫脩)

从不同的参数值创建一个数组。 内存消耗是一样的 `uniqExact` 功能。

第二个版本（与 `max_size` 参数）将结果数组的大小限制为 `max_size` 元素。

例如, `groupUniqArray(1)(x)` 相当于 `[any(x)]`.

分位数

计算近似值 **分位数** 的数字数据序列。

此功能适用 **油藏采样** 随着储存器大小高达8192和随机数发生器进行采样。 结果是非确定性的。 要获得精确的分位数，请使用 `quantileExact` 功能。

当使用多个 `quantile*` 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。 在这种情况下，使用 **分位数** 功能。

语法

```
quantile(level)(expr)
```

别名: `median`.

参数

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` 值的范围 [0.01, 0.99]. 默认值：0.5。 在 `level=0.5` 该函数计算 **中位数**.
- `expr` — Expression over the column values resulting in numeric 数据类型, 日期 或 日期时间.

返回值

- 指定电平的近似分位数。

类型:

- **Float64** 对于数字数据类型输入。
- **日期** 如果输入值具有 **Date** 类型。
- **日期时间** 如果输入值具有 **DateTime** 类型。

示例

输入表:

val
1
1
2
3

查询:

```
SELECT quantile(val) FROM t
```

结果:

quantile(val)
1.5

另请参阅

- **中位数**
- **分位数**

量化确定

计算近似值 **分位数** 的数字数据序列。

此功能适用 **油藏采样** 与储层大小高达**8192**和采样的确定性算法。结果是确定性的。要获得精确的分位数，请使用 **quantileExact** 功能。

当使用多个 **quantile*** 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 **分位数** 功能。

语法

```
quantileDeterministic(level)(expr, determinator)
```

别名: `medianDeterministic.`

参数

- **level** — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a **level** 值的范围 [0.01, 0.99]. 默认值：0.5。在 **level=0.5** 该函数计算 **中位数**。
- **expr** — Expression over the column values resulting in numeric **数据类型**, **日期** 或 **日期时间**.
- **determinator** — Number whose hash is used instead of a random number generator in the reservoir sampling algorithm to make the result of sampling deterministic. As a determinator you can use any deterministic positive number, for example, a user id or an event id. If the same determinator value occurs too often the function works incorrectly

occurred too often, the function works incorrectly.

返回值

- 指定电平的近似分位数。

类型:

- Float64** 对于数字数据类型输入。
- 日期** 如果输入值具有 **Date** 类型。
- 日期时间** 如果输入值具有 **DateTime** 类型。

示例

输入表:

val
1
1
2
3

查询:

```
SELECT quantileDeterministic(val, 1) FROM t
```

结果:

quantileDeterministic(val, 1)
1.5

另请参阅

- 中位数**
- 分位数**

quantileExact

正是计算 **分位数** 的数字数据序列。

To get exact value, all the passed values are combined into an array, which is then partially sorted.

Therefore, the function consumes **O(n)** 内存，其中 **n** 是传递的多个值。然而，对于少量的值，该函数是非常有效的。

当使用多个 **quantile*** 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 **分位数** 功能。

语法

```
quantileExact(level)(expr)
```

别名: **medianExact**.

参数

- level** — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a **level** 值的范围 [0.01, 0.99]. 默认值：0.5。在 **level=0.5** 该函数计算 **中位数**。

- `expr` — Expression over the column values resulting in numeric 数据类型, 日期 或 日期时间.

返回值

- 指定电平的分位数。

类型:

- `Float64` 对于数字数据类型输入。
- `日期` 如果输入值具有 `Date` 类型。
- `日期时间` 如果输入值具有 `DateTime` 类型。

示例

查询:

```
SELECT quantileExact(number) FROM numbers(10)
```

结果:

```
└─quantileExact(number)─
   └─5 ─
```

另请参阅

- 中位数
- 分位数

分位数加权

正是计算 `分位数` 数值数据序列，考虑到每个元素的权重。

To get exact value, all the passed values are combined into an array, which is then partially sorted. Each value is counted with its weight, as if it is present `weight` times. A hash table is used in the algorithm. Because of this, if the passed values are frequently repeated, the function consumes less RAM than `quantileExact`. 您可以使用此功能，而不是 `quantileExact` 并指定重量1。

当使用多个 `quantile*` 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 `分位数` 功能。

语法

```
quantileExactWeighted(level)(expr, weight)
```

别名: `medianExactWeighted`.

参数

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` 值的范围 [0.01, 0.99]. 默认值：0.5。在 `level=0.5` 该函数计算 `中位数`.
- `expr` — Expression over the column values resulting in numeric 数据类型, 日期 或 日期时间.
- `weight` — Column with weights of sequence members. Weight is a number of value occurrences.

返回值

- 指定电平的分位数。

类型:

- **Float64** 对于数字数据类型输入。
- **日期** 如果输入值具有 **Date** 类型。
- **日期时间** 如果输入值具有 **DateTime** 类型。

示例

输入表:

n	val
0	3
1	2
2	1
5	4

查询:

```
SELECT quantileExactWeighted(n, val) FROM t
```

结果:

quantileExactWeighted(n, val)
1

另请参阅

- **中位数**
- **分位数**

分位定时

随着确定的精度计算 **分位数** 的数字数据序列。

结果是确定性的（它不依赖于查询处理顺序）。该函数针对描述加载网页时间或后端响应时间等分布的序列进行了优化。

当使用多个 **quantile*** 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 **分位数** 功能。

语法

```
quantileTiming(level)(expr)
```

别名: **medianTiming**.

参数

- **level** — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a **level** 值的范围 [0.01, 0.99]. 默认值：0.5。在 **level=0.5** 该函数计算 **中位数**。
- **expr** — 表达式 在一个列值返回 **浮动*-键入号码**。

- If negative values are passed to the function, the behavior is undefined.
- If the value is greater than 30,000 (a page loading time of more than 30 seconds), it is assumed to be 30,000.

精度

计算是准确的，如果：

- 值的总数不超过5670。
- 总数值超过5670，但页面加载时间小于1024ms。

否则，计算结果将四舍五入到16毫秒的最接近倍数。

注

对于计算页面加载时间分位数，此函数比 分位数.

返回值

- 指定电平的分位数。

类型: `Float32`.

注

如果没有值传递给函数（当使用 `quantileTimingIf`），`NaN` 被返回。这样做的目的是将这些案例与导致零的案例区分开来。看 [按条款订购 对于排序注意事项](#) `NaN` 值。

示例

输入表:

```
└─response_time─|
    72 |
    112 |
    126 |
    145 |
    104 |
    242 |
    313 |
    168 |
    108 |
```

查询:

```
SELECT quantileTiming(response_time) FROM t
```

结果:

```
└─quantileTiming(response_time)─|
    126 |
```

另请参阅

- [中位数](#)
- [分位数](#)

分位时间加权

随着确定的精度计算 分位数 根据每个序列成员的权重对数字数据序列进行处理。

结果是确定性的（它不依赖于查询处理顺序）。该函数针对描述加载网页时间或后端响应时间等分布的序列进行了优化。

当使用多个 `quantile*` 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 分位数 功能。

语法

```
quantileTimingWeighted(level)(expr, weight)
```

别名: `medianTimingWeighted`.

参数

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` 值的范围 [0.01, 0.99]. 默认值：0.5。在 `level=0.5` 该函数计算 中位数.
- `expr` — 表达式 在一个列值返回 浮动*-键入号码。

- If negative values are passed to the function, the behavior is undefined.
- If the value is greater than 30,000 (a page loading time of more than 30 seconds), it is assumed to be 30,000.

- `weight` — Column with weights of sequence elements. Weight is a number of value occurrences.

精度

计算是准确的，如果：

- 值的总数不超过5670。
- 总数值超过5670，但页面加载时间小于1024ms。

否则，计算结果将四舍五入到16毫秒的最接近倍数。

注

对于计算页面加载时间分位数，此函数比 分位数.

返回值

- 指定电平的分位数。

类型: `Float32`.

注

如果没有值传递给函数（当使用 `quantileTimingIf`），阿南 被返回。这样做的目的是将这些案例与导致零的案例区分开来。看 按条款订购 对于排序注意事项 `Nan` 值。

示例

输入表:

response_time	weight
68	1

104	2
112	3
126	2
138	1
162	1

查询:

```
SELECT quantileTimingWeighted(response_time, weight) FROM t
```

结果:

quantileTimingWeighted(response_time, weight)
112

另请参阅

- 中位数
- 分位数

quantileTDigest

计算近似值 分位数 使用的数字数据序列 **t-digest** 算法。

最大误差为 1%。 内存消耗 $\log(n)$ ，哪里 n 是多个值。 结果取决于运行查询的顺序，并且是不确定的。

该功能的性能低于性能 分位数 或 分位定时。在状态大小与精度的比率方面，这个函数比 **quantile**。

当使用多个 **quantile*** 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 分位数 功能。

语法

```
quantileTDigest(level)(expr)
```

别名: **medianTDigest**.

参数

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` 值的范围 [0.01, 0.99]。默认值：0.5。在 `level=0.5` 该函数计算 中位数.
- `expr` — Expression over the column values resulting in numeric 数据类型, 日期 或 日期时间.

回值

- 指定电平的近似分位数。

类型:

- **Float64** 对于数字数据类型输入。
- **日期** 如果输入值具有 `Date` 类型。
- **日期时间** 如果输入值具有 `DateTime` 类型。

示例

查询:

```
SELECT quantileTDigest(number) FROM numbers(10)
```

结果:

```
└─quantileTDigest(number)─  
    4.5 |
```

另请参阅

- 中位数
- 分位数

quantileTDigestWeighted

计算近似值 分位数 使用的数字数据序列 **t-digest** 算法。该函数考虑了每个序列成员的权重。最大误差为 1%。内存消耗 $\log(n)$ ，哪里 n 是多个值。

该功能的性能低于性能 分位数 或 分位定时。在状态大小与精度的比率方面，这个函数比 `quantile`。

结果取决于运行查询的顺序，并且是不确定的。

当使用多个 `quantile*` 在查询中具有不同级别的函数，内部状态不会被组合（即查询的工作效率低于它可以）。在这种情况下，使用 分位数 功能。

语法

```
quantileTDigest(level)(expr)
```

别名: `medianTDigest`。

参数

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` 值的范围 [0.01, 0.99]。默认值：0.5。在 `level=0.5` 该函数计算 中位数。
- `expr` — Expression over the column values resulting in numeric 数据类型, 日期 或 日期时间。
- `weight` — Column with weights of sequence elements. Weight is a number of value occurrences.

返回值

- 指定电平的近似分位数。

类型:

- **Float64** 对于数字数据类型输入。
- **日期** 如果输入值具有 `Date` 类型。
- **日期时间** 如果输入值具有 `DateTime` 类型。

示例

查询:

```
SELECT quantileTDigestWeighted(number, 1) FROM numbers(10)
```

结果:

```
└─quantileTDigestWeighted(number, 1)─  
    4.5 |
```

另请参阅

- 中位数
- 分位数

中位数

该 `median*` 函数是相应的别名 `quantile*` 功能。它们计算数字数据样本的中位数。

功能:

- `median` — Alias for 分位数.
- `medianDeterministic` — Alias for 量化确定.
- `medianExact` — Alias for `quantileExact`.
- `medianExactWeighted` — Alias for 分位数加权.
- `medianTiming` — Alias for 分位定时.
- `medianTimingWeighted` — Alias for 分位时间加权.
- `medianTDigest` — Alias for `quantileTDigest`.
- `medianTDigestWeighted` — Alias for `quantileTDigestWeighted`.

示例

输入表:

```
└─val─  
  | 1 |  
  | 1 |  
  | 2 |  
  | 3 |
```

查询:

```
SELECT medianDeterministic(val, 1) FROM t
```

结果:

```
└─medianDeterministic(val, 1)─  
    1.5 |
```

quantiles(level1, level2, ...)(x)

所有分位数函数也具有相应的分位数函数: `quantiles`, `quantilesDeterministic`, `quantilesTiming`, `quantilesTimingWeighted`, `quantilesExact`, `quantilesExactWeighted`, `quantilesTDigest`. 这些函数在一遍中计算所列电平的所有分位数，并返回结果值的数组。

varSamp(x)

计算金额 $\sum((x - \bar{x})^2) / (n - 1)$ ，哪里 `n` 是样本大小和 `\bar{x}` 是平均值 `x`.

它表示随机变量的方差的无偏估计，如果传递的值形成其样本。

返回 `Float64` 当 `n >= 1`，返回 `NaN`

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 `varSampStable` 功能。它的工作速度较慢，但提供较低的计算错误。

`varPop(x)`

计算金额 $\sum((x - \bar{x})^2) / n$ ，哪里 `n` 是样本大小和 `\bar{x}` 是平均值 `x`。

换句话说，分散为一组值。返回 `Float64`。

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 `varPopStable` 功能。它的工作速度较慢，但提供较低的计算错误。

`stddevSamp(x)`

结果等于平方根 `varSamp(x)`。

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 `stddevSampStable` 功能。它的工作速度较慢，但提供较低的计算错误。

`stddevPop(x)`

结果等于平方根 `varPop(x)`。

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 `stddevPopStable` 功能。它的工作速度较慢，但提供较低的计算错误。

`topK(N)(x)`

返回指定列中近似最常见值的数组。生成的数组按值的近似频率降序排序（而不是值本身）。

实现了 过滤节省空间 基于reduce-and-combine算法的TopK分析算法 并行节省空间。

`topK(N)(column)`

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常见值的常见值。

我们建议使用 `N < 10` 值；性能降低了大 `N` 值。的最大值 `N = 65536`。

参数

- ‘`N`’ 是要返回的元素数。

如果省略该参数，则使用默认值10。

参数

- 'x' – The value to calculate frequency.

示例

就拿 **时间** 数据集，并选择在三个最频繁出现的值 **AirlineID** 列。

```
SELECT topK(3)(AirlineID) AS res
FROM ontime
```

```
res
[19393,19790,19805]
```

topKWeighted

类似于 **topK** 但需要一个整数类型的附加参数 - **weight**. 每个价值都被记入 **weight** 次频率计算。

语法

```
topKWeighted(N)(x, weight)
```

参数

- **N** — The number of elements to return.

参数

- **x** – The value.
- **weight** — The weight. **UInt8**.

返回值

返回具有最大近似权重总和的值数组。

示例

查询:

```
SELECT topKWeighted(10)(number, number) FROM numbers(1000)
```

结果:

```
topKWeighted(10)(number, number)
[999,998,997,996,995,994,993,992,991,990]
```

covarSamp(x,y)

计算的值 $\sum((x - \bar{x})(y - \bar{y})) / (n - 1)$.

返回 **Float64**。当 **n <= 1**, returns $+\infty$.

注

该函数使用数值不稳定的算法。如果你需要 **数值稳定性** 在计算中，使用 **covarSampStable** 功能。它的工作速度较慢，但提供较低的计算错误。

covarPop(x,y)

计算的值 $\Sigma((x - \bar{x})(y - \bar{y})) / n$.

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 covarPopStable 功能。它的工作速度较慢，但提供了较低的计算错误。

corr(x,y)

计算Pearson相关系数: $\Sigma((x - \bar{x})(y - \bar{y})) / \sqrt{\Sigma((x - \bar{x})^2) * \Sigma((y - \bar{y})^2)}$.

注

该函数使用数值不稳定的算法。如果你需要 数值稳定性 在计算中，使用 corrStable 功能。它的工作速度较慢，但提供较低的计算错误。

categoricallInformationValue

计算的值 $(P(tag = 1) - P(tag = 0))(\log(P(tag = 1)) - \log(P(tag = 0)))$ 对于每个类别。

```
categoricallInformationValue(category1, category2, ..., tag)
```

结果指示离散（分类）要素如何使用 [category1, category2, ...] 有助于预测的价值的学习模型 tag.

simpleLinearRegression

执行简单（一维）线性回归。

```
simpleLinearRegression(x, y)
```

参数:

- x — Column with dependent variable values.
- y — Column with explanatory variable values.

返回值:

常量 (a, b) 结果行的 $y = a*x + b$.

例

```
SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3])
```

```
| arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3]) |  
| (1,0) |
```

```
SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6])
```

```
|arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6])|  
|(1,3)|
```

随机指标线上回归

该函数实现随机线性回归。它支持自定义参数的学习率，L2正则化系数，迷你批量大小，并具有更新权重的方法很少（**应当**（默认使用），**简单 SGD**，**动量**，**Nesterov**）。

参数

有4个可自定义的参数。它们按顺序传递给函数，但是没有必要传递所有四个默认值将被使用，但是好的模型需要一些参数调整。

```
stochasticLinearRegression(1.0, 1.0, 10, 'SGD')
```

1. **learning rate** 当执行梯度下降步骤时，步长上的系数。过大的学习率可能会导致模型的权重无限大。默认值为 **0.00001**.
2. **L2 regularization coefficient** 这可能有助于防止过度拟合。默认值为 **0.1**.
3. **mini-batch size** 设置元素的数量，这些元素将被计算和求和以执行梯度下降的一个步骤。纯随机下降使用一个元素，但是具有小批量（约10个元素）使梯度步骤更稳定。默认值为 **15**.
4. **method for updating weights** 他们是：**Adam**（默认情况下），**SGD**，**Momentum**，**Nesterov**. **Momentum** 和 **Nesterov** 需要更多的计算和内存，但是它们恰好在收敛速度和随机梯度方法的稳定性方面是有用的。

用途

stochasticLinearRegression 用于两个步骤：拟合模型和预测新数据。为了拟合模型并保存其状态以供以后使用，我们使用 **-State combinator**，它基本上保存了状态（模型权重等）。

为了预测我们使用函数 **evalMLMethod**，这需要一个状态作为参数以及特征来预测。

1. 适合

可以使用这种查询。

```
CREATE TABLE IF NOT EXISTS train_data  
(  
    param1 Float64,  
    param2 Float64,  
    target Float64  
) ENGINE = Memory;  
  
CREATE TABLE your_model ENGINE = Memory AS SELECT  
stochasticLinearRegressionState(0.1, 0.0, 5, 'SGD')(target, param1, param2)  
AS state FROM train_data;
```

在这里，我们还需要将数据插入到 **train_data** 桌子 参数的数量不是固定的，它只取决于参数的数量，传递到 **linearRegressionState**。它们都必须是数值。

请注意，带有目标值的列（我们想要学习预测）被插入作为第一个参数。

2. 预测

在将状态保存到表中之后，我们可以多次使用它进行预测，甚至与其他状态合并并创建新的更好的模型。

```
WITH (SELECT state FROM your_model) AS model SELECT  
evalMLMethod(model, param1, param2) FROM test_data
```

查询将返回一列预测值。请注意，第一个参数 `evalMLMethod` 是 `AggregateFunctionState` 对象，接下来是要素列。

`test_data` 是一个像表 `train_data` 但可能不包含目标值。

注

- 要合并两个模型，用户可以创建这样的查询：

```
sql SELECT state1 + state2 FROM your_models
```

哪里 `your_models` 表包含这两个模型。此查询将返回 `new AggregateFunctionState` 对象。

- 如果没有，用户可以获取创建的模型的权重用于自己的目的，而不保存模型 `-State` 使用 `combinator`。

```
sql SELECT stochasticLinearRegression(0.01)(target, param1, param2) FROM train_data
```

这种查询将拟合模型并返回其权重-首先是权重，它对应于模型的参数，最后一个偏差。所以在上面的例子中，查询将返回一个具有3个值的列。

另请参阅

- [stochasticLogisticRegression](#)
- [线性回归和逻辑回归之间的区别](#)

stochasticLogisticRegression

该函数实现随机逻辑回归。它可以用于二进制分类问题，支持与 `stochasticLinearRegression` 相同的自定义参数，并以相同的方式工作。

参数

参数与 `stochasticLinearRegression` 中的参数完全相同：

`learning rate`, `l2 regularization coefficient`, `mini-batch size`, `method for updating weights`.

欲了解更多信息，请参阅 [参数](#)。

```
stochasticLogisticRegression(1.0, 1.0, 10, 'SGD')
```

- 适合

See the `Fitting` section in the [\[stochasticLinearRegression\]\(#stochasticlinearregression-usage-fitting\)](#) description.

Predicted labels have to be in `\[-1, 1\]`.

- 预测

Using saved state we can predict probability of object having label `1`.

```
```sql
WITH (SELECT state FROM your_model) AS model SELECT
evalMLMethod(model, param1, param2) FROM test_data
```

```

The query will return a column of probabilities. Note that first argument of `evalMLMethod` is `AggregateFunctionState` object, next are columns of features.

We can also set a bound of probability, which assigns elements to different labels.

```
```sql
SELECT ans < 1.1 AND ans > 0.5 FROM
(WITH (SELECT state FROM your_model) AS model SELECT
evalMLMethod(model, param1, param2) AS ans FROM test_data)
```

```

Then the result will be labels.

`test_data` is a table like `train_data` but may not contain target value.

另请参阅

- 随机指标线上回归
- 线性回归和逻辑回归之间的差异。

groupBitmapAnd

计算位图列的AND，返回 UInt64类型的基数，如果添加后缀状态，则返回 位图对象.

groupBitmapAnd(expr)

参数

expr – An expression that results in `AggregateFunction(groupBitmap, UInt*)` 类型。

返回值

的值 `UInt64` 类型。

示例

```
DROP TABLE IF EXISTS bitmap_column_expr_test2;
CREATE TABLE bitmap_column_expr_test2
(
    tag_id String,
    z AggregateFunction(groupBitmap, UInt32)
)
ENGINE = MergeTree
ORDER BY tag_id;

INSERT INTO bitmap_column_expr_test2 VALUES ('tag1', bitmapBuild(cast([1,2,3,4,5,6,7,8,9,10] as Array(UInt32))));
INSERT INTO bitmap_column_expr_test2 VALUES ('tag2', bitmapBuild(cast([6,7,8,9,10,11,12,13,14,15] as
Array(UInt32))));
INSERT INTO bitmap_column_expr_test2 VALUES ('tag3', bitmapBuild(cast([2,4,6,8,10,12] as Array(UInt32))));

SELECT groupBitmapAnd(z) FROM bitmap_column_expr_test2 WHERE like(tag_id, 'tag%');
└groupBitmapAnd(z)┘
  3 |
```

```
SELECT arraySort(bitmapToArray(groupBitmapAndState(z))) FROM bitmap_column_expr_test2 WHERE like(tag_id,
'tag%');
└arraySort(bitmapToArray(groupBitmapAndState(z)))┘
  [6,8,10] |
```

groupBitmapOr

计算位图列的OR，返回 UInt64类型的基数，如果添加后缀状态，则返回 位图对象. 这相当于 `groupBitmapMerge`.

groupBitmapOr(expr)

参数

expr – An expression that results in `AggregateFunction(groupBitmap, UInt*)` 类型。

`expr` – An expression that results in `AggregateFunction(groupBitmap, UInt*)` 类型。

返回值

的值 `UInt64` 类型。

示例

```
DROP TABLE IF EXISTS bitmap_column_expr_test2;
CREATE TABLE bitmap_column_expr_test2
(
    tag_id String,
    z AggregateFunction(groupBitmap, UInt32)
)
ENGINE = MergeTree
ORDER BY tag_id;

INSERT INTO bitmap_column_expr_test2 VALUES ('tag1', bitmapBuild(cast([1,2,3,4,5,6,7,8,9,10] as Array(UInt32))));  
INSERT INTO bitmap_column_expr_test2 VALUES ('tag2', bitmapBuild(cast([6,7,8,9,10,11,12,13,14,15] as  
Array(UInt32))));  
INSERT INTO bitmap_column_expr_test2 VALUES ('tag3', bitmapBuild(cast([2,4,6,8,10,12] as Array(UInt32))));

SELECT groupBitmapOr(z) FROM bitmap_column_expr_test2 WHERE like(tag_id, 'tag%');
└groupBitmapOr(z)─
   15 |
```



```
SELECT arraySort(bitmapToArray(groupBitmapOrState(z))) FROM bitmap_column_expr_test2 WHERE like(tag_id, 'tag%');
└arraySort(bitmapToArray(groupBitmapOrState(z)))─
 [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] |
```

groupBitmapXor

计算位图列的XOR，返回 `UInt64` 类型的基数，如果添加后缀状态，则返回 [位图对象](#)。

`groupBitmapOr(expr)`

参数

`expr` – An expression that results in `AggregateFunction(groupBitmap, UInt*)` 类型。

返回值

的值 `UInt64` 类型。

示例

```
DROP TABLE IF EXISTS bitmap_column_expr_test2;
CREATE TABLE bitmap_column_expr_test2
(
    tag_id String,
    z AggregateFunction(groupBitmap, UInt32)
)
ENGINE = MergeTree
ORDER BY tag_id;

INSERT INTO bitmap_column_expr_test2 VALUES ('tag1', bitmapBuild(cast([1,2,3,4,5,6,7,8,9,10] as Array(UInt32))));  
INSERT INTO bitmap_column_expr_test2 VALUES ('tag2', bitmapBuild(cast([6,7,8,9,10,11,12,13,14,15] as  
Array(UInt32))));
```

```
INSERT INTO bitmap_column_expr_test2 VALUES ('tag3', bitmapBuild(cast([2,4,6,8,10,12] as Array(UInt32))));
```

```
SELECT groupBitmapXor(z) FROM bitmap_column_expr_test2 WHERE like(tag_id, 'tag%');
```

```
└─groupBitmapXor(z)─
```

```
  | 10 |
```

```
SELECT arraySort(bitmapToArray(groupBitmapXorState(z))) FROM bitmap_column_expr_test2 WHERE like(tag_id, 'tag%');
```

```
└─arraySort(bitmapToArray(groupBitmapXorState(z)))─
```

```
  | [1,3,5,6,8,10,11,13,14,15] |
```

聚合函数组合器

聚合函数的名称可以附加一个后缀。这改变了聚合函数的工作方式。

-如果

The suffix `-If` can be appended to the name of any aggregate function. In this case, the aggregate function accepts an extra argument – a condition (UInt8 type). The aggregate function processes only the rows that trigger the condition. If the condition was not triggered even once, it returns a default value (usually zeros or empty strings).

例: `sumIf(column, cond)`, `countIf(cond)`, `avgIf(x, cond)`, `quantilesTimingIf(level1, level2)(x, cond)`, `argMinIf(arg, val, cond)` 等等。

使用条件聚合函数，您可以一次计算多个条件的聚合，而无需使用子查询和 `JOIN` 例如，在 Yandex 的 Metrica，条件聚合函数用于实现段比较功能。

-阵列

`-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。

示例1: `sumArray(arr)` - 总计所有的所有元素 ‘`arr`’ 阵列。在这个例子中，它可以更简单地编写: `sum(arraySum(arr))`.

示例2: `uniqArray(arr)` – Counts the number of unique elements in all ‘`arr`’ 阵列。这可以做一个更简单的方法: `uniq(arrayJoin(arr))`，但它并不总是可以添加 ‘`arrayJoin`’ 到查询。

-如果和-阵列可以组合。然而，‘`Array`’ 必须先来，然后 ‘`If`’。例: `uniqArrayIf(arr, cond)`, `quantilesTimingArrayIf(level1, level2)(arr, cond)`. 由于这个顺序，该 ‘`cond`’ 参数不会是数组。

-州

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` 函数），但聚合的中间状态（用于 `uniq`，这是用于计算唯一值的数量的散列表）。这是一个 `AggregateFunction(...)` 可用于进一步处理或存储在表中以完成聚合。

要使用这些状态，请使用:

- `AggregatingMergeTree` 表引擎。
- 最后聚会 功能。
- 跑累积 功能。
- -合并 `combinator`
- -`MergeState` `combinator`

合并

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 combinator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 combinator。

-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combinator，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。

语法

```
<aggFunction>OrDefault(x)
```

参数

- `x` — Aggregate function parameters.

返回值

如果没有要聚合的内容，则返回聚合函数返回类型的默认值。

类型取决于所使用的聚合函数。

示例

查询:

```
SELECT avg(number), avgOrDefault(number) FROM numbers(0)
```

结果:

```
avg(number) avgOrDefault(number)
nan          0
```

还有 `-OrDefault` 可与其他组合器一起使用。当聚合函数不接受空输入时，它很有用。

查询:

```
SELECT avgOrDefaultIf(x, x > 10)
FROM
(
  SELECT toDecimal32(1.23, 2) AS x
)
```

结果:

```
└─avgOrDefaultIf(x, greater(x, 10))─  
    0.00 |
```

-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 数据类型。如果聚合函数没有值来计算它返回 **NULL**。

`-OrNull` 可与其他组合器一起使用。

语法

```
<aggFunction>OrNull(x)
```

参数

- `x` — Aggregate function parameters.

返回值

- 聚合函数的结果，转换为 `Nullable` 数据类型。
- `NULL`，如果没有进行任何聚合。

类型: `Nullable(aggregate function return type)`.

示例

添加 `-orNull` 到聚合函数的末尾。

查询:

```
SELECT sumOrNull(number), toTypeName(sumOrNull(number)) FROM numbers(10) WHERE number > 10
```

结果:

```
└─sumOrNull(number)─┘ toTypeName(sumOrNull(number))─  
    NULL | Nullable(UInt64) |
```

还有 `-OrNone` 可与其他组合器一起使用。当聚合函数不接受空输入时，它很有用。

查询:

```
SELECT avgOrNoneIf(x, x > 10)  
FROM  
(  
    SELECT toDecimal32(1.23, 2) AS x  
)
```

结果:

```
└─avgOrNoneIf(x, greater(x, 10))─  
    NULL |
```

- 重新采样

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。

```
<aggFunction>Resample(start, end, step)(<aggFunction_params>, resampling_key)
```

参数

- `start` — Starting value of the whole required interval for `resampling_key` 值。
- `stop` — Ending value of the whole required interval for `resampling_key` 值。整个时间间隔不包括 `stop` 价值 `[start, stop)`。
- `step` — Step for separating the whole interval into subintervals. The `aggFunction` 在每个子区间上独立执行。
- `resampling_key` — Column whose values are used for separating data into intervals.
- `aggFunction_params` — `aggFunction` 参数。

返回值

- 阵列 `aggFunction` 每个子区间的結果。

示例

考虑一下 `people` 具有以下数据的表:

| name | age | wage |
|--------|-----|------|
| John | 16 | 10 |
| Alice | 30 | 15 |
| Mary | 35 | 8 |
| Evelyn | 48 | 11.5 |
| David | 62 | 9.9 |
| Brian | 60 | 16 |

让我们得到的人的名字，他们的年龄在于的时间间隔 `[30,60)` 和 `[60,75)`。由于我们使用整数表示的年龄，我们得到的年龄 `[30, 59]` 和 `[60,74]` 间隔。

要在数组中聚合名称，我们使用 `groupArray` 聚合函数。这需要一个参数。在我们的例子中，它是 `name` 列。该 `groupArrayResample` 函数应该使用 `age` 按年龄聚合名称的列。要定义所需的时间间隔，我们通过 `30, 75, 30` 参数到 `groupArrayResample` 功能。

```
SELECT groupArrayResample(30, 75, 30)(name, age) FROM people
```

```
groupArrayResample(30, 75, 30)(name, age)
[[['Alice','Mary','Evelyn'],['David','Brian']] ]
```

考虑结果。

`John` 是因为他太年轻了 其他人按照指定的年龄间隔进行分配。

现在让我们计算指定年龄间隔内的总人数和平均工资。

```
SELECT
countResample(30, 75, 30)(name, age) AS amount,
avgResample(30, 75, 30)(wage, name) AS avg_wage
```

```
avgResample(30, 10, 30)(wage, age) AS avg_wage  
FROM people
```

```
amount avg_wage  
| [3,2] | [11.5,12.949999809265137] |
```

参数聚合函数

Some aggregate functions can accept not only argument columns (used for compression), but a set of parameters – constants for initialization. The syntax is two pairs of brackets instead of one. The first is for parameters, and the second is for arguments.

直方图

计算自适应直方图。它不能保证精确的结果。

```
histogram(number_of_bins)(values)
```

该函数使用 [流式并行决策树算法](#)。当新数据输入函数时，hist图分区的边界将被调整。在通常情况下，箱的宽度不相等。

参数

`number_of_bins` — Upper limit for the number of bins in the histogram. The function automatically calculates the number of bins. It tries to reach the specified number of bins, but if it fails, it uses fewer bins.

`values` — 表达式 导致输入值。

返回值

- 阵列 的 元组 下面的格式:

```
...  
[(lower_1, upper_1, height_1), ... (lower_N, upper_N, height_N)]  
...  
  
- `lower` — Lower bound of the bin.  
- `upper` — Upper bound of the bin.  
- `height` — Calculated height of the bin.
```

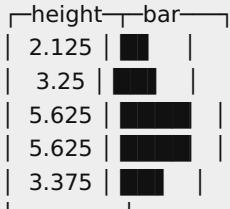
示例

```
SELECT histogram(5)(number + 1)  
FROM (  
    SELECT *  
    FROM system.numbers  
    LIMIT 20  
)
```

```
histogram(5)(plus(number, 1))  
| [(1,4.5,4),(4.5,8.5,4),(8.5,12.75,4.125),(12.75,17,4.625),(17,20,3.25)] |
```

您可以使用 [酒吧](#) 功能，例如：

```
WITH histogram(5)(rand() % 100) AS hist
SELECT
    arrayJoin(hist).3 AS height,
    bar(height, 0, 6, 5) AS bar
FROM
(
    SELECT *
    FROM system.numbers
    LIMIT 20
)
```



在这种情况下，您应该记住您不知道直方图bin边界。

sequenceMatch(pattern)(timestamp, cond1, cond2, ...)

检查序列是否包含与模式匹配的事件链。

```
sequenceMatch(pattern)(timestamp, cond1, cond2, ...)
```

警告

在同一秒钟发生的事件可能以未定义的顺序排列在序列中，影响结果。

参数

- `pattern` — Pattern string. See [模式语法](#).
- `timestamp` — Column considered to contain time data. Typical data types are `Date` 和 `DateTime`. 您还可以使用任何支持的 `UInt` 数据类型。
- `cond1, cond2` — Conditions that describe the chain of events. Data type: `UInt8`. 最多可以传递32个条件参数。该函数只考虑这些条件下描述的事件。如果序列包含未在条件下描述的数据，则函数将跳过这些数据。

返回值

- 1，如果模式匹配。
- 0，如果模式不匹配。

类型: `UInt8`.

模式语法

- `(?N)` — Matches the condition argument at position `N`. 条件在编号 [1, 32] 范围。例如, `(?1)` 匹配传递给 `cond1` 参数。
- `.*` — Matches any number of events. You don't need conditional arguments to match this element of the pattern.

- `(?t operator value)` — Sets the time in seconds that should separate two events. For example, pattern `(?1)(?t>1800)(?2)` 匹配彼此发生超过1800秒的事件。这些事件之间可以存在任意数量的任何事件。您可以使用 `>=`, `>`, `<`, `<=` 运营商。

例

考虑在数据 `t` 表:

| time | number |
|------|--------|
| 1 | 1 |
| 2 | 3 |
| 3 | 2 |

执行查询:

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceMatch('(?1)(?2)')(time, equals(number, 1), equals(number, 2))—
    1 |
```

该函数找到了数字2跟随数字1的事件链。它跳过了它们之间的数字3，因为该数字没有被描述为事件。如果我们想在搜索示例中给出的事件链时考虑这个数字，我们应该为它创建一个条件。

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2, number = 3) FROM t
```

```
sequenceMatch('(?1)(?2)')(time, equals(number, 1), equals(number, 2), equals(number, 3))—
    0 |
```

在这种情况下，函数找不到与模式匹配的事件链，因为数字3的事件发生在1和2之间。如果在相同的情况下，我们检查了数字4的条件，则序列将与模式匹配。

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2, number = 4) FROM t
```

```
sequenceMatch('(?1)(?2)')(time, equals(number, 1), equals(number, 2), equals(number, 4))—
    1 |
```

另请参阅

- [sequenceCount](#)

sequenceCount(pattern)(time, cond1, cond2, ...)

计数与模式匹配的事件链的数量。该函数搜索不重叠的事件链。当前链匹配后，它开始搜索下一个链。

警告

在同一秒钟发生的事件可能以未定义的顺序排列在序列中，影响结果。

```
sequenceCount(pattern)(timestamp, cond1, cond2, ...)
```

参数

- `pattern` — Pattern string. See [模式语法](#).
- `timestamp` — Column considered to contain time data. Typical data types are `Date` 和 `DateTime`. 您还可以使用任何支持的 `UInt` 数据类型。
- `cond1, cond2` — Conditions that describe the chain of events. Data type: `UInt8`. 最多可以传递32个条件参数。该函数只考虑这些条件下描述的事件。如果序列包含未在条件下描述的数据，则函数将跳过这些数据。

返回值

- 匹配的非重叠事件链数。

类型: `UInt64`.

示例

考虑在数据 `t` 表:

| time | number |
|------|--------|
| 1 | 1 |
| 2 | 3 |
| 3 | 2 |
| 4 | 1 |
| 5 | 3 |
| 6 | 2 |

计算数字2在数字1之后出现的次数以及它们之间的任何其他数字:

```
SELECT sequenceCount('(?1).*(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceCount('(?1).*(?2)')(time, equals(number, 1), equals(number, 2))
```

2 |

另请参阅

- [sequenceMatch](#)

windowFunnel

搜索滑动时间窗中的事件链，并计算从链中发生的最大事件数。

该函数根据算法工作:

- 该函数搜索触发链中的第一个条件并将事件计数器设置为1的数据。这是滑动窗口启动的时刻。
- 如果来自链的事件在窗口内顺序发生，则计数器将递增。如果事件序列中断，则计数器不会增加。
- 如果数据在不同的完成点具有多个事件链，则该函数将仅输出最长链的大小。

语法

```
windowFunnel(window, [mode])(timestamp, cond1, cond2, ..., condN)
```

参数

- `window` — Length of the sliding window in seconds.
- `mode` - 这是一个可选的参数。
 - 'strict' - 当 'strict' 设置时，`windowFunnel()` 仅对唯一值应用条件。
- `timestamp` — Name of the column containing the timestamp. Data types supported: 日期, 日期时间 和其他无符号整数类型 (请注意，即使时间戳支持 `UInt64` 类型，它的值不能超过 `Int64` 最大值，即 $2^{63}-1$)。
- `cond` — Conditions or data describing the chain of events. `UInt8`.

返回值

滑动时间窗口内连续触发条件链的最大数目。

对选择中的所有链进行了分析。

类型: `Integer`.

示例

确定设定的时间段是否足以让用户选择手机并在在线商店中购买两次。

设置以下事件链:

1. 用户登录到其在应用商店中的帐户 (`eventID = 1003`).
2. 用户搜索手机 (`eventID = 1007, product = 'phone'`).
3. 用户下了订单 (`eventID = 1009`).
4. 用户再次下订单 (`eventID = 1010`).

输入表:

| event_date | user_id | timestamp | eventID | product |
|------------|---------|---------------------|---------|---------|
| 2019-01-28 | 1 | 2019-01-29 10:00:00 | 1003 | phone |
| 2019-01-31 | 1 | 2019-01-31 09:00:00 | 1007 | phone |
| 2019-01-30 | 1 | 2019-01-30 08:00:00 | 1009 | phone |
| 2019-02-01 | 1 | 2019-02-01 08:00:00 | 1010 | phone |

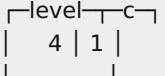
了解用户有多远 `user_id` 可以在 2019 的 1-2 月期间通过链条。

查询:

```
SELECT
    level,
    count() AS c
FROM
(
    SELECT
        user_id,
        windowFunnel(6048000000000000)(timestamp, eventID = 1003, eventID = 1009, eventID = 1007, eventID = 1010)
AS level
    FROM trend
    WHERE (event_date >= '2019-01-01') AND (event_date <= '2019-02-02')
```

```
    GROUP BY user_id  
)  
GROUP BY level  
ORDER BY level ASC
```

结果:



保留

该函数将一组条件作为参数，类型为1到32个参数 `UInt8` 表示事件是否满足特定条件。

任何条件都可以指定为参数（如 **WHERE**）。

除了第一个以外，条件成对适用：如果第一个和第二个是真的，第二个结果将是真的，如果第一个和第三个是真的，第三个结果将是真的，等等。

语法

```
retention(cond1, cond2, ..., cond32);
```

参数

- `cond` — an expression that returns a `UInt8` 结果（1或0）。

返回值

数组为1或0。

- 1 — condition was met for the event.
- 0 — condition wasn't met for the event.

类型: `UInt8`.

示例

让我们考虑计算的一个例子 `retention` 功能，以确定网站流量。

1. Create a table to illustrate an example.

```
CREATE TABLE retention_test(date Date, uid Int32) ENGINE = Memory;  
  
INSERT INTO retention_test SELECT '2020-01-01', number FROM numbers(5);  
INSERT INTO retention_test SELECT '2020-01-02', number FROM numbers(10);  
INSERT INTO retention_test SELECT '2020-01-03', number FROM numbers(15);
```

输入表:

查询:

```
SELECT * FROM retention_test
```

结果:

| date | uid |
|------------|-----|
| 2020-01-01 | 0 |
| 2020-01-01 | 1 |
| 2020-01-01 | 2 |
| 2020-01-01 | 3 |
| 2020-01-01 | 4 |

| date | uid |
|------------|-----|
| 2020-01-02 | 0 |
| 2020-01-02 | 1 |
| 2020-01-02 | 2 |
| 2020-01-02 | 3 |
| 2020-01-02 | 4 |
| 2020-01-02 | 5 |
| 2020-01-02 | 6 |
| 2020-01-02 | 7 |
| 2020-01-02 | 8 |
| 2020-01-02 | 9 |

| date | uid |
|------------|-----|
| 2020-01-03 | 0 |
| 2020-01-03 | 1 |
| 2020-01-03 | 2 |
| 2020-01-03 | 3 |
| 2020-01-03 | 4 |
| 2020-01-03 | 5 |
| 2020-01-03 | 6 |
| 2020-01-03 | 7 |
| 2020-01-03 | 8 |
| 2020-01-03 | 9 |
| 2020-01-03 | 10 |
| 2020-01-03 | 11 |
| 2020-01-03 | 12 |
| 2020-01-03 | 13 |
| 2020-01-03 | 14 |

2. 按唯一ID对用户进行分组 uid 使用 retention 功能。

查询:

```
SELECT
    uid,
    retention(date = '2020-01-01', date = '2020-01-02', date = '2020-01-03') AS r
FROM retention_test
WHERE date IN ('2020-01-01', '2020-01-02', '2020-01-03')
GROUP BY uid
ORDER BY uid ASC
```

结果:

| uid | r |
|-----|---------|
| 0 | [1,1,1] |
| 1 | [1,1,1] |
| 2 | [1,1,1] |
| 3 | [1,1,1] |
| 4 | [1,1,1] |
| 5 | [0,0,0] |
| 6 | [0,0,0] |
| 7 | [0,0,0] |

| | |
|----|---------|
| 7 | [0,0,0] |
| 8 | [0,0,0] |
| 9 | [0,0,0] |
| 10 | [0,0,0] |
| 11 | [0,0,0] |
| 12 | [0,0,0] |
| 13 | [0,0,0] |
| 14 | [0,0,0] |

3. 计算每天的现场访问总数。

查询:

```

SELECT
    sum(r[1]) AS r1,
    sum(r[2]) AS r2,
    sum(r[3]) AS r3
FROM
(
    SELECT
        uid,
        retention(date = '2020-01-01', date = '2020-01-02', date = '2020-01-03') AS r
    FROM retention_test
    WHERE date IN ('2020-01-01', '2020-01-02', '2020-01-03')
    GROUP BY uid
)

```

结果:

| | | |
|----|----|----|
| r1 | r2 | r3 |
| 5 | 5 | 5 |

哪里:

- r1 - 2020-01-01期间访问该网站的独立访问者数量（cond1 条件）。
- r2 - 在2020-01-01和2020-01-02之间的特定时间段内访问该网站的唯一访问者的数量（cond1 和 cond2 条件）。
- r3 - 在2020-01-01和2020-01-03之间的特定时间段内访问该网站的唯一访问者的数量（cond1 和 cond3 条件）。

uniqUpTo(N)(x)

Calculates the number of different argument values if it is less than or equal to N. If the number of different argument values is greater than N, it returns N + 1.

建议使用小Ns，高达10。N的最大值为100。

对于聚合函数的状态，它使用的内存量等于 $1+N$ *一个字节值的大小。

对于字符串，它存储8个字节的非加密哈希。也就是说，计算是近似的字符串。

该函数也适用于多个参数。

它的工作速度尽可能快，除了使用较大的N值并且唯一值的数量略小于N的情况。

用法示例:

Problem: Generate a report that shows only keywords that produced at least 5 unique users.
 Solution: Write in the GROUP BY query SearchPhrase HAVING uniqUpTo(4)(UserID) >= 5

sumMapFiltered(keys_to_keep)(键值)

同样的行为 `sumMap` 除了一个键数组作为参数传递。这在使用高基数密钥时尤其有用。

表函数

表函数是构造表的方法。

您可以使用表函数：

- `FROM` 《公约》条款 `SELECT` 查询。

The method for creating a temporary table that is available only in the current query. The table is deleted when the query finishes.

- 创建表为\<table_function()> 查询。

It's one of the methods of creating a table.

警告

你不能使用表函数，如果 `allow_ddl` 设置被禁用。

| 功能 | 产品描述 |
|-------|---|
| 文件 | 创建一个 <code>文件</code> -发动机表。 |
| 合并 | 创建一个 <code>合并</code> -发动机表。 |
| 数字 | 创建一个包含整数填充的单列的表。 |
| 远程 | 允许您访问远程服务器，而无需创建 <code>分布</code> -发动机表。 |
| url | 创建一个 <code>Url</code> -发动机表。 |
| mysql | 创建一个 <code>MySQL</code> -发动机表。 |
| jdbc | 创建一个 <code>JDBC</code> -发动机表。 |
| odbc | 创建一个 <code>ODBC</code> -发动机表。 |
| hdfs | 创建一个 <code>HDFS</code> -发动机表。 |

文件

从文件创建表。此表函数类似于 `url` 和 `hdfs` 一些的。

`file(path, format, structure)`

输入参数

- `path` — The relative path to the file from `user_files_path`. 只读模式下的`glob`s后的文件支持路径: `*`, `?`, `{abc,def}` 和 `{N..M}` 哪里 `N, M` — numbers, `'abc', 'def'` — strings.
- `format` — The 格式 的文件。
- `structure` — Structure of the table. Format `'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

设置 `user_files_path` 和文件的内容 `test.csv`:

```
$ grep user_files_path /etc/clickhouse-server/config.xml
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>

$ cat /var/lib/clickhouse/user_files/test.csv
1,2,3
3,2,1
78,43,45
```

表从 `test.csv` 并从中选择前两行:

```
SELECT *
FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32')
LIMIT 2
```

| column1 | column2 | column3 |
|---------|---------|---------|
| 1 | 2 | 3 |
| 3 | 2 | 1 |

```
-- getting the first 10 lines of a table that contains 3 columns of UInt32 type from a CSV file
SELECT * FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32') LIMIT 10
```

路径中的水珠

多个路径组件可以具有`glob`s。对于正在处理的文件应该存在并匹配到整个路径模式（不仅后缀或前缀）。

- `*` — Substitutes any number of any characters except / 包括空字符串。
- `?` — Substitutes any single character.
- `{some_string,another_string,yet_another_one}` — Substitutes any of strings `'some_string', 'another_string', 'yet_another_one'`.
- `{N..M}` — Substitutes any number in range from N to M including both borders.

建筑与 `{}` 类似于 [远程表功能](#)).

示例

1. 假设我们有几个具有以下相对路径的文件:

- `'some_dir/some_file_1'`
- `'some_dir/some_file_2'`
- `'some_dir/some_file_3'`
- `'another_dir/some_file_1'`

- `another_dir/some_file_*`
- 'another_dir/some_file_2'
- 'another_dir/some_file_3'

1. 查询这些文件中的行数:

```
SELECT count(*)
FROM file('{some,another}_dir/some_file_{1..3}', 'TSV', 'name String, value UInt32')
```

1. 查询这两个目录的所有文件中的行数:

```
SELECT count(*)
FROM file('{some,another}_dir/*', 'TSV', 'name String, value UInt32')
```

警告

如果您的文件列表包含带前导零的数字范围，请单独使用带大括号的构造或使用 `?`.

示例

从名为 `file000, file001, ..., file999`:

```
SELECT count(*)
FROM file('big_dir/file{0..9}{0..9}{0..9}', 'CSV', 'name String, value UInt32')
```

虚拟列

- `_path` — Path to the file.
- `_file` — Name of the file.

另请参阅

- [虚拟列](#)

合并

`merge(db_name, 'tables_regexp')` – Creates a temporary Merge table. For more information, see the section “Table engines, Merge”.

表结构取自与正则表达式匹配的第一个表。

数字

`numbers(N)` – Returns a table with the single ‘number’ 包含从0到N-1的整数的列(UInt64)。
`numbers(N, M)` -返回一个表与单 ‘number’ 包含从N到(N+M-1)的整数的列(UInt64)。

类似于 `system.numbers` 表，它可以用于测试和生成连续的值，`numbers(N, M)` 比 `system.numbers`.

以下查询是等效的:

```
SELECT * FROM numbers(10);
SELECT * FROM numbers(0, 10);
```

```
SELECT * FROM system.numbers LIMIT 10;
```

例：

```
-- Generate a sequence of dates from 2010-01-01 to 2010-12-31
select toDate('2010-01-01') + number as d FROM numbers(365);
```

远程，远程安全

允许您访问远程服务器，而无需创建 **Distributed** 桌子

签名：

```
remote('addresses_expr', db, table[, 'user'][, 'password']])
remote('addresses_expr', db.table[, 'user'][, 'password'])
```

addresses_expr – An expression that generates addresses of remote servers. This may be just one server address. The server address is `host:port`，或者只是 `host`. 主机可以指定为服务器名称，也可以指定为IPv4或IPv6地址。IPv6地址在方括号中指定。端口是远程服务器上的TCP端口。如果省略端口，它使用 `tcp_port` 从服务器的配置文件（默认情况下，9000）。

重要事项

IPv6地址需要该端口。

例：

```
example01-01-1
example01-01-1:9000
localhost
127.0.0.1
[::]:9000
[2a02:6b8:0:1111::11]:9000
```

多个地址可以用逗号分隔。在这种情况下，ClickHouse将使用分布式处理，因此它将将查询发送到所有指定的地址（如具有不同数据的分片）。

示例：

```
example01-01-1,example01-02-1
```

表达式的一部分可以用大括号指定。前面的示例可以写成如下：

```
example01-0{1,2}-1
```

大括号可以包含由两个点（非负整数）分隔的数字范围。在这种情况下，范围将扩展为生成分片地址的一组值。如果第一个数字以零开头，则使用相同的零对齐形成值。前面的示例可以写成如下：

```
example01-{01..02}-1
```

如果您有多对大括号，它会生成相应集合的直接乘积。

大括号中的地址和部分地址可以用管道符号(|)分隔。在这种情况下，相应的地址集被解释为副本，并且查询将被发送到第一个正常副本。但是，副本将按照当前设置的顺序进行迭代 **load_balancing** 设置。

示例：

```
example01-{01..02}-{1|2}
```

此示例指定两个分片，每个分片都有两个副本。

生成的地址数由常量限制。现在这是1000个地址。

使用 `remote` 表函数比创建一个不太优化 `Distributed` 表，因为在这种情况下，服务器连接被重新建立为每个请求。此外，如果设置了主机名，则会解析这些名称，并且在使用各种副本时不会计算错误。在处理大量查询时，始终创建 `Distributed` 表的时间提前，不要使用 `remote` 表功能。

该 `remote` 表函数可以在以下情况下是有用的：

- 访问特定服务器进行数据比较、调试和测试。
- 查询之间的各种ClickHouse群集用于研究目的。
- 手动发出的罕见分布式请求。
- 每次重新定义服务器集的分布式请求。

如果未指定用户，`default` 被使用。

如果未指定密码，则使用空密码。

`remoteSecure` -相同 `remote` but with secured connection. Default port — **tcp_port_secure** 从配置或9440.

url

`url(URL, format, structure)` -返回从创建的表 `URL` 与给定 `format` 和 `structure`.

URL-HTTP或HTTPS服务器地址，它可以接受 `GET` 和/或 `POST` 请求。

格式 - **格式** 的数据。

结构-表结构 '`UserID UInt64, Name String`' 格式。确定列名称和类型。

示例

```
-- getting the first 3 lines of a table that contains columns of String and UInt32 type from HTTP-server which answers in CSV format.  
SELECT * FROM url('http://127.0.0.1:12345/', CSV, 'column1 String, column2 UInt32') LIMIT 3
```

mysql

允许 `SELECT` 要对存储在远程MySQL服务器上的数据执行的查询。

```
mysql('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause']);
```

参数

- `host:port` — MySQL server address.
- `database` — Remote database name.
- `table` — Remote table name.
- `user` — MySQL user.
- `password` — User password.
- `replace_query` — Flag that converts `INSERT INTO` 查询到 `REPLACE INTO`. 如果 `replace_query=1`，查询被替换。
- `on_duplicate_clause` — The `ON DUPLICATE KEY on_duplicate_clause` 表达式被添加到 `INSERT` 查询。

Example: `INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1` , where `on_duplicate_clause` is `UPDATE c2 = c2 + 1` . See the MySQL documentation to find which `on_duplicate_clause` you can use with the `ON DUPLICATE KEY` clause.

To specify `on_duplicate_clause` you need to pass `0` to the `replace_query` parameter. If you simultaneously pass `replace_query = 1` and `on_duplicate_clause` , ClickHouse generates an exception.

简单 `WHERE` 条款如 `=, !=, >, >=, <, <=` 当前在MySQL服务器上执行。

其余的条件和 `LIMIT` 只有在对MySQL的查询完成后，才会在ClickHouse中执行采样约束。

返回值

与原始MySQL表具有相同列的table对象。

用法示例

MySQL中的表:

```
mysql> CREATE TABLE `test`.`test` (
->   `int_id` INT NOT NULL AUTO_INCREMENT,
->   `int_nullable` INT NULL DEFAULT NULL,
->   `float` FLOAT NOT NULL,
->   `float_nullable` FLOAT NULL DEFAULT NULL,
->   PRIMARY KEY (`int_id`));
Query OK, 0 rows affected (0,09 sec)

mysql> insert into test (`int_id`, `float`) VALUES (1,2);
Query OK, 1 row affected (0,00 sec)

mysql> select * from test;
+-----+-----+-----+
| int_id | int_nullable | float | float_nullable |
+-----+-----+-----+
|    1 |        NULL |    2 |        NULL |
+-----+-----+-----+
1 row in set (0,00 sec)
```

从ClickHouse中选择数据:

```
SELECT * FROM mysql('localhost:3306', 'test', 'test', 'bayonet', '123')
```

| int_id | int_nullable | float | float_nullable |
|--------|--------------|-------|----------------|
| 1 | NULL | 2 | NULL |

另请参阅

- 该 ‘MySQL’ 表引擎
- 使用MySQL作为外部字典的来源

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` 程序正在运行。

它支持可空类型（基于查询的远程表的DDL）。

例

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

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

```
SELECT * FROM jdbc('datasource://mysql-local', 'schema', 'table')
```

odbc

返回通过连接的表 ODBC。

```
odbc(connection_settings, external_database, external_table)
```

参数:

- `connection_settings` — Name of the section with connection settings in the `odbc.ini` 文件
- `external_database` — Name of a database in an external DBMS.
- `external_table` — Name of a table in the `external_database`.

为了安全地实现ODBC连接，ClickHouse使用单独的程序 `clickhouse-odbc-bridge`。如果直接从ODBC驱动程序加载 `clickhouse-server`，驱动程序问题可能会导致ClickHouse服务器崩溃。ClickHouse自动启动 `clickhouse-odbc-bridge` 当它是必需的。ODBC桥程序是从相同的软件包作为安装 `clickhouse-server`。

与字段 `NULL` 外部表中的值将转换为基数据类型的默认值。例如，如果远程MySQL表字段具有 `INT NULL` 键入它将转换为 0 (ClickHouse的默认值 `Int32` 数据类型)。

用法示例

通过ODBC从本地MySQL安装获取数据

此示例检查Ubuntu Linux18.04和MySQL服务器5.7。

确保安装了unixODBC和MySQL连接器。

默认情况下（如果从软件包安装），ClickHouse以用户身份启动 `clickhouse`。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;
```

然后配置连接 `/etc/odbc.ini`.

```
$ cat /etc/odbc.ini
[mysqlconn]
DRIVER = /usr/local/lib/libmyodbc5w.so
SERVER = 127.0.0.1
PORT = 3306
DATABASE = test
USERNAME = clickhouse
PASSWORD = clickhouse
```

您可以使用 `isql unixodbc` 安装中的实用程序。

```
$ isql -v mysqlconn
+-----+
| Connected!
| |
...|
```

MySQL中的表:

```
mysql> CREATE TABLE `test`.`test` (
-> `int_id` INT NOT NULL AUTO_INCREMENT,
-> `int_nullable` INT NULL DEFAULT NULL,
-> `float` FLOAT NOT NULL,
-> `float_nullable` FLOAT NULL DEFAULT NULL,
-> PRIMARY KEY (`int_id`));
Query OK, 0 rows affected (0,09 sec)
```

```
mysql> insert into test (`int_id`, `float`) VALUES (1,2);
Query OK, 1 row affected (0,00 sec)
```

```
mysql> select * from test;
+-----+-----+-----+
| int_id | int_nullable | float | float_nullable |
+-----+-----+-----+
|    1 |      NULL |    2 |      NULL |
+-----+-----+-----+
1 row in set (0,00 sec)
```

从ClickHouse中的MySQL表中检索数据:

```
SELECT * FROM odbc('DSN=mysqlconn', 'test', 'test')
```

| int_id | int_nullable | float | float_nullable |
|--------|--------------|-------|----------------|
| 1 | 0 | 2 | 0 |

另请参阅

- [ODBC外部字典](#)
- [ODBC表引擎](#).

hdfs

从HDFS中的文件创建表。此表函数类似于 [url](#) 和 [文件](#) 一些的。

```
hdfs(URI, format, structure)
```

输入参数

- `URI` — The relative URI to the file in HDFS. Path to file support following globs in readonly mode: `*`, `?`, `{abc,def}` 和 `{N..M}` 哪里 `N, M` — numbers, `'abc', 'def'` — strings.
- `format` — The [格式](#) 的文件。
- `structure` — Structure of the table. Format `'column1_name column1_type, column2_name column2_type, ...'`.

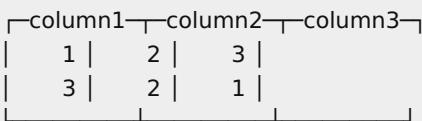
返回值

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

表从 `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
```



路径中的水珠

多个路径组件可以具有globs。对于正在处理的文件应该存在并匹配到整个路径模式（不仅后缀或前缀）。

- `*` — Substitutes any number of any characters except / 包括空字符串。
- `?` — Substitutes any single character.
- `{some_string,another_string,yet_another_one}` — Substitutes any of strings `'some_string', 'another_string', 'yet_another_one'`.
- `{N..M}` — Substitutes any number in range from N to M including both borders.

建筑与 `{}` 类似于 [远程表功能](#)).

示例

1. 假设我们在HDFS上有几个具有以下Uri的文件:

- `'hdfs://hdfs1:9000/some_dir/some_file_1'`
- `'hdfs://hdfs1:9000/some_dir/some_file_2'`
- `'hdfs://hdfs1:9000/some_dir/some_file_3'`
- `'hdfs://hdfs1:9000/another_dir/some_file_1'`

- 'hdfs://hdfs1:9000/another_dir/some_file_2'
- 'hdfs://hdfs1:9000/another_dir/some_file_3'

1. 查询这些文件中的行数:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/{some,another}_dir/some_file_{1..3}', 'TSV', 'name String, value UInt32')
```

1. 查询这两个目录的所有文件中的行数:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/{some,another}_dir/*', 'TSV', 'name String, value UInt32')
```

警告

如果您的文件列表包含带前导零的数字范围，请单独使用带大括号的构造或使用 `?`.

示例

从名为 `file000, file001, ..., file999`:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/big_dir/file{0..9}{0..9}{0..9}', 'CSV', 'name String, value UInt32')
```

虚拟列

- `_path` — Path to the file.
- `_file` — Name of the file.

另请参阅

- [虚拟列](#)

输入

`input(structure)` - 表功能，允许有效地转换和插入数据发送到服务器与给定结构的表与另一种结构。

`structure` - 以下格式发送到服务器的数据结构 `'column1_name column1_type, column2_name column2_type, ...'`. 例如, `'id UInt32, name String'`.

此功能只能用于 `INSERT SELECT` 查询，只有一次，但其他行为像普通表函数(例如，它可以用于子查询等。).

数据可以以任何方式像普通发送 `INSERT` 查询并传递任何可用 [格式](#) 必须在查询结束时指定 (不像普通 `INSERT SELECT`).

这个功能的主要特点是，当服务器从客户端接收数据时，它同时将其转换根据表达式中的列表 `SELECT` 子句并插入到目标表中。临时表不创建所有传输的数据。

例

- 让 `test` 表具有以下结构 `(a String, b String)`

和数据 `data.csv` 具有不同的结构 (`col1 String, col2 Date, col3 Int32`). 查询插入从数据 `data.csv` 进 `test` 同时转换的表如下所示:

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

- 如果 `data.csv` 包含相同结构的数据 `test_structure` 作为表 `test` 那么这两个查询是相等的:

```
$ cat data.csv | clickhouse-client --query="INSERT INTO test FORMAT CSV"  
$ cat data.csv | clickhouse-client --query="INSERT INTO test SELECT * FROM input('test_structure') FORMAT CSV"
```

generateRandom

使用给定的模式生成随机数据。

允许用数据填充测试表。

支持可以存储在表中的所有数据类型，除了 `LowCardinality` 和 `AggregateFunction`.

```
generateRandom('name TypeName[, name TypeName]...', [, 'random_seed'[, 'max_string_length'[,  
'max_array_length']]])
```

参数

- `name` — Name of corresponding column.
- `TypeName` — Type of corresponding column.
- `max_array_length` — Maximum array length for all generated arrays. Defaults to `10`.
- `max_string_length` — Maximum string length for all generated strings. Defaults to `10`.
- `random_seed` — Specify random seed manually to produce stable results. If `NONE` — seed is randomly generated.

返回值

具有请求架构的表对象。

用法示例

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

| a | d | c |
|----------|--------------|--|
| [77] | -124167.6723 | ('2061-04-17 21:59:44.573','3f72f405-ec3e-13c8-44ca-66ef335f7835') |
| [32,110] | -141397.7312 | ('1979-02-09 03:43:48.526','982486d1-5a5d-a308-e525-7bd8b80ffa73') |
| [68] | -67417.0770 | ('2080-03-12 14:17:31.269','110425e5-413f-10a6-05ba-fa6b3e929f15') |

字典

字典是一个映射 (`key -> attributes`) 这是方便各种类型的参考清单。

ClickHouse 支持使用可用于查询的字典的特殊功能。这是更容易和更有效地使用字典与功能比 `JOIN` 与参考表。

`NULL` 值不能存储在字典中。

ClickHouse支持:

- 内置字典 具有特定的 功能集.
- 插件（外部）字典 用一个 功能集.

外部字典

您可以从各种数据源添加自己的字典。字典的数据源可以是本地文本或可执行文件、HTTP(s)资源或其他DBMS。有关详细信息，请参阅“[外部字典的来源](#)”。

ClickHouse:

- 完全或部分存储在RAM中的字典。
- 定期更新字典并动态加载缺失的值。换句话说，字典可以动态加载。
- 允许创建外部字典与xml文件或 [DDL查询](#).

外部字典的配置可以位于一个或多个xml文件中。配置的路径在指定 [dictionaries_config](#) 参数。

字典可以在服务器启动或首次使用时加载，具体取决于 [dictionaries_lazy_load](#) 设置。

该 [字典](#) 系统表包含有关在服务器上配置的字典的信息。对于每个字典，你可以在那里找到:

- 字典的状态。
- 配置参数。
- 度量指标，如为字典分配的RAM量或自成功加载字典以来的查询数量。

字典配置文件具有以下格式:

```
<yandex>
<comment>An optional element with any content. Ignored by the ClickHouse server.</comment>

<!--Optional element. File name with substitutions-->
<include_from>/etc/metrika.xml</include_from>

<dictionary>
  <!-- Dictionary configuration. -->
  <!-- There can be any number of <dictionary> sections in the configuration file. -->
</dictionary>

</yandex>
```

你可以 [配置](#) 同一文件中的任意数量的字典。

[字典的DDL查询](#) 在服务器配置中不需要任何其他记录。它们允许使用字典作为一流的实体，如表或视图。

注意

您可以通过在一个小字典中描述它来转换小字典的值 [SELECT](#) 查询（见 [变换功能](#)）。此功能与外部字典无关。

另请参阅

- [配置外部字典](#)
- [在内存中存储字典](#)
- [字典函数](#)

- **丁六文列**
- 外部字典的来源
- 字典键和字段
- 使用外部字典的函数

配置外部字典

如果使用 XML 文件配置字典，则字典配置具有以下结构：

```
<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>
```

相应的 **DDL-查询** 具有以下结构：

```
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
```

- **name** — The identifier that can be used to access the dictionary. Use the characters [a-zA-Z0-9_\-].
- **来源** — Source of the dictionary.
- **布局** — Dictionary layout in memory.
- **结构** — Structure of the dictionary . A key and attributes that can be retrieved by this key.
- **使用寿命** — Frequency of dictionary updates.

在内存中存储字典

有多种方法可以将字典存储在内存中。

我们建议 **平**, **散列** 和 **complex_key_hashed**. 其提供最佳的处理速度。

不建议使用缓存，因为性能可能较差，并且难以选择最佳参数。阅读更多的部分“**缓存**”。

有几种方法可以提高字典性能：

- 调用该函数以使用后的字典 `GROUP BY`.
- 将要提取的属性标记为"注射"。如果不同的属性值对应于不同的键，则称为注射属性。所以当 `GROUP BY` 使用由键获取属性值的函数，此函数会自动取出 `GROUP BY`.

ClickHouse为字典中的错误生成异常。错误示例：

- 无法加载正在访问的字典。
- 查询错误 `cached` 字典

您可以查看外部字典的列表及其状态 `system.dictionaries` 桌子

配置如下所示：

```
<yandex>
  <dictionary>
    ...
    <layout>
      <layout_type>
        <!-- layout settings -->
      </layout_type>
    </layout>
    ...
  </dictionary>
</yandex>
```

相应的 **DDL-查询**：

```
CREATE DICTIONARY (...)

...
LAYOUT(LAYOUT_TYPE(param value)) -- layout settings
...
```

在内存中存储字典的方法

- 平
- 散列
- `sparse_hashed`
- 缓存
- 直接
- `range_hashed`
- `complex_key_hashed`
- `complex_key_cache`
- `ip_trie`

平

字典以平面数组的形式完全存储在内存中。字典使用多少内存？量与最大键的大小（在使用的空间中）成正比。

字典键具有 `UInt64` 类型和值限制为 500,000。如果在创建字典时发现较大的键，ClickHouse将引发异常，不会创建字典。

支持所有类型的来源。更新时，数据（来自文件或表）将完整读取。

此方法在存储字典的所有可用方法中提供了最佳性能。

配置示例：

```
<layout>
  <flat />
```

```
</layout>
```

或

```
LAYOUT(FLAT())
```

散列

该字典以哈希表的形式完全存储在内存中。字典中可以包含任意数量的带有任意标识符的元素，在实践中，键的数量可以达到数千万项。

支持所有类型的来源。更新时，数据（来自文件或表）将完整读取。

配置示例：

```
<layout>
<hashed />
</layout>
```

或

```
LAYOUT(HASHED())
```

sparse_hashed

类似于 `hashed`，但使用更少的内存，有利于更多的CPU使用率。

配置示例：

```
<layout>
<sparse_hashed />
</layout>
```

```
LAYOUT(SPARSE_HASHED())
```

complex_key_hashed

这种类型的存储是用于复合 **键**。类似于 `hashed`。

配置示例：

```
<layout>
<complex_key_hashed />
</layout>
```

```
LAYOUT(COMPLEX_KEY_HASHED())
```

range_hashed

字典以哈希表的形式存储在内存中，其中包含有序范围及其相应值的数组。

此存储方法的工作方式与散列方式相同，除了键之外，还允许使用日期/时间（任意数字类型）范围。

示例：该表格包含每个广告客户的折扣，格式为：

advertiser id	discount start date	discount end date	amount
123	2015-01-01	2015-01-15	0.15
123	2015-01-16	2015-01-31	0.25
456	2015-01-01	2015-01-15	0.05

要对日期范围使用示例，请定义 `range_min` 和 `range_max` 中的元素 **结构**。这些元素必须包含元素 `name` 和 `type`（如果 `type` 没有指定，则默认类型将使用 `-Date`）。`type` 可以是任何数字类型（`Date/DateTime/UInt64/Int32/others`）。

示例：

```
<structure>
<id>
  <name>Id</name>
</id>
<range_min>
  <name>first</name>
  <type>Date</type>
</range_min>
<range_max>
  <name>last</name>
  <type>Date</type>
</range_max>
...
...
```

或

```
CREATE DICTIONARY somedict (
    id UInt64,
    first Date,
    last Date
)
PRIMARY KEY id
LAYOUT(RANGE_HASHED())
RANGE(MIN first MAX last)
```

要使用这些字典，您需要将附加参数传递给 `dictGetT` 函数，为其选择一个范围：

```
dictGetT('dict_name', 'attr_name', id, date)
```

此函数返回指定的值 `ids` 和包含传递日期的日期范围。

算法的详细信息：

- 如果 `id` 未找到或范围未找到 `id`，它返回字典的默认值。
- 如果存在重叠范围，则可以使用任意范围。
- 如果范围分隔符是 `NULL` 或无效日期（如 `1900-01-01` 或 `2039-01-01`），范围保持打开状态。范围可以在两侧打开。

配置示例：

```
<yandex>
```

```

<dictionary>
...
<layout>
    <range_hashed />
</layout>

<structure>
    <id>
        <name>Abcdef</name>
    </id>
    <range_min>
        <name>StartTimeStamp</name>
        <type>UInt64</type>
    </range_min>
    <range_max>
        <name>EndTimeStamp</name>
        <type>UInt64</type>
    </range_max>
    <attribute>
        <name>XXXType</name>
        <type>String</type>
        <null_value />
    </attribute>
</structure>

</dictionary>
</yandex>

```

或

```

CREATE DICTIONARY somedict(
    Abcdef UInt64,
    StartTimeStamp UInt64,
    EndTimeStamp UInt64,
    XXXType String DEFAULT ""
)
PRIMARY KEY Abcdef
RANGE(MIN StartTimeStamp MAX EndTimeStamp)

```

缓存

字典存储在具有固定数量的单元格的缓存中。这些单元格包含经常使用的元素。

搜索字典时，首先搜索缓存。对于每个数据块，所有在缓存中找不到或过期的密钥都从源请求，使用 `SELECT attrs... FROM db.table WHERE id IN (k1, k2, ...)`。然后将接收到的数据写入高速缓存。

对于缓存字典，过期 [使用寿命](#) 可以设置高速缓存中的数据。如果更多的时间比 `lifetime` 自从在单元格中加载数据以来，单元格的值不被使用，并且在下次需要使用时重新请求它。

这是存储字典的所有方法中最不有效的。缓存的速度在很大程度上取决于正确的设置和使用场景。缓存类型字典只有在命中率足够高（推荐99%或更高）时才能表现良好。您可以查看平均命中率 `system.dictionaries` 桌子

要提高缓存性能，请使用以下子查询 `LIMIT`，并从外部调用字典函数。

支持 [来源](#):MySQL的,ClickHouse的,可执行文件,HTTP.

设置示例:

```
<layout>
```

```
<cache>
  <!-- The size of the cache, in number of cells. Rounded up to a power of two. -->
  <size_in_cells>1000000000</size_in_cells>
</cache>
</layout>
```

或

```
LAYOUT(CACHE(SIZE_IN_CELLS 1000000000))
```

设置足够大的缓存大小。你需要尝试选择细胞的数量：

1. 设置一些值。
2. 运行查询，直到缓存完全满。
3. 使用评估内存消耗 `system.dictionaries` 桌子
4. 增加或减少单元数，直到达到所需的内存消耗。

警告

不要使用ClickHouse作为源，因为处理随机读取的查询速度很慢。

complex_key_cache

这种类型的存储是用于复合 [键](#)，类似于 `cache`。

直接

字典不存储在内存中，并且在处理请求期间直接转到源。

字典键具有 `UInt64` 类型。

所有类型的 [来源](#)，除了本地文件，支持。

配置示例：

```
<layout>
  <direct />
</layout>
```

或

```
LAYOUT(DIRECT())
```

ip_trie

这种类型的存储用于将网络前缀（IP地址）映射到ASN等元数据。

示例：该表包含网络前缀及其对应的AS号码和国家代码：

prefix	asn	cca2
202.79.32.0/20	17501	NP
2620:0:870::/48	3856	US
2a02:6b8:1::/48	13238	RU

```
+-----+-----+
| 2001:db8::/32 | 65536 | ZZ   |
+-----+-----+
```

使用此类布局时，结构必须具有复合键。

示例：

```
<structure>
<key>
  <attribute>
    <name>prefix</name>
    <type>String</type>
  </attribute>
</key>
<attribute>
  <name>asn</name>
  <type>UInt32</type>
  <null_value />
</attribute>
<attribute>
  <name>cca2</name>
  <type>String</type>
  <null_value>??</null_value>
</attribute>
...
...
```

或

```
CREATE DICTIONARY somedict (
  prefix String,
  asn UInt32,
  cca2 String DEFAULT '??'
)
PRIMARY KEY prefix
```

该键必须只有一个包含允许的IP前缀的字符串类型属性。还不支持其他类型。

对于查询，必须使用相同的函数 (`dictGetT` 与元组) 至于具有复合键的字典：

```
dictGetT('dict_name', 'attr_name', tuple(ip))
```

该函数采用任一 `UInt32` 对于IPv4，或 `FixedString(16)` 碌莽禄Ipv6拢IPv6：

```
dictGetString('prefix', 'asn', tuple(IPv6StringToNum('2001:db8::1')))
```

还不支持其他类型。该函数返回与此IP地址对应的前缀的属性。如果有重叠的前缀，则返回最具体的前缀。

数据存储在一个 `trie`。它必须完全适合RAM。

字典更新

ClickHouse定期更新字典。完全下载字典的更新间隔和缓存字典的无效间隔在 `<lifetime>` 在几秒钟内标记。

字典更新（除首次使用的加载之外）不会阻止查询。在更新期间，将使用旧版本的字典。如果在更新过程中发生错误，则将错误写入服务器日志，并使用旧版本的字典继续查询。

设置示例：

```
<dictionary>
...
<lifetime>300</lifetime>
...
</dictionary>
```

```
CREATE DICTIONARY (...)
```

```
...
LIFETIME(300)
...
```

设置 `<lifetime>0</lifetime>` (`LIFETIME(0)`) 防止字典更新。

您可以设置升级的时间间隔，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。

设置示例：

```
<dictionary>
...
<lifetime>
  <min>300</min>
  <max>360</max>
</lifetime>
...
</dictionary>
```

或

```
LIFETIME(MIN 300 MAX 360)
```

如果 `<min>0</min>` 和 `<max>0</max>`，ClickHouse不会按超时重新加载字典。

在这种情况下，如果字典配置文件已更改，ClickHouse可以更早地重新加载字典 `SYSTEM RELOAD DICTIONARY` 命令被执行。

升级字典时，ClickHouse服务器根据字典的类型应用不同的逻辑 [来源](#)：

升级字典时，ClickHouse服务器根据字典的类型应用不同的逻辑 [来源](#)：

- 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](#)。

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'))
```

外部字典的来源

外部字典可以从许多不同的来源连接。

如果使用 `xml-file` 配置字典，则配置如下所示：

```
<yandex>
<dictionary>
...
<source>
  <source_type>
    <!-- Source configuration -->
  </source_type>
</source>
...
</dictionary>
...
</yandex>
```

在情况下 **DDL-查询**，相等的配置将看起来像：

```
CREATE DICTIONARY dict_name (...)

...
SOURCE(SOURCE_TYPE(param1 val1 ... paramN valN)) -- Source configuration
...
```

源配置在 `source` 科。

对于源类型 **本地文件**, **可执行文件**, **HTTP(s)**, **ClickHouse**

可选设置：

```
<source>
<file>
  <path>/opt/dictionaries/os.tsv</path>
  <format>TabSeparated</format>
</file>
<settings>
  <format_csv_allow_single_quotes>0</format_csv_allow_single_quotes>
</settings>
```

```
</source>
```

或

```
SOURCE(FILE(path '/opt/dictionaries/os.tsv' format 'TabSeparated'))  
SETTINGS(format_csv_allow_single_quotes = 0)
```

来源类型 (`source_type`):

- 本地文件
- 可执行文件
- [HTTP\(s\)](#)
- DBMS
 - [ODBC](#)
 - [MySQL](#)
 - [ClickHouse](#)
 - [MongoDB](#)
 - [Redis](#)

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 '/opt/dictionaries/os.tsv' format 'TabSeparated'))
```

设置字段:

- `path` – The absolute path to the file.
- `format` – The file format. All the formats described in “[格式](#)” 支持。

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](#). 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'))
```

设置字段:

- `command` – The absolute path to the executable file, or the file name (if the program directory is written to `PATH`).
- `format` – The file format. All the formats described in “[格式](#)” 支持。

Http(s)

使用HTTP (s) 服务器取决于 [字典如何存储在内存中](#). 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](#) 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](#)” 支持。
- `credentials` – Basic HTTP authentication. Optional parameter.
 - `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

您可以使用此方法连接具有ODBC驱动程序的任何数据库。

设置示例:

```
<source>
<odbc>
  <db>DatabaseName</db>
  <table>SchemaName.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'
))
```

设置字段:

- db – Name of the database. Omit it if the database name is set in the `<connection_string>` parameter.
- table – Name of the table and schema if exists.
- connection_string – Connection string.
- invalidate_query – Query for checking the dictionary status. Optional parameter. Read more in the section [更新字典](#).

ClickHouse接收来自ODBC-driver的引用符号，并将查询中的所有设置引用到driver，因此有必要根据数据库中的表名大小写设置表名。

如果您在使用Oracle时遇到编码问题，请参阅相应的 [FAQ](#) 文章。

ODBC字典功能的已知漏洞

注意

通过ODBC驱动程序连接参数连接到数据库时 `Servername` 可以取代。在这种情况下，值 `USERNAME` 和 `PASSWORD` 从 `odbc.ini` 被发送到远程服务器，并且可能会受到损害。

不安全使用示例

让我们为PostgreSQL配置unixODBC。的内容 `/etc/odbc.ini`:

```
[gregtest]
Driver = /usr/lib/pgsqlodbc.so
Servername = localhost
PORT = 5432
DATABASE = test_db
##OPTION = 3
USERNAME = test
PASSWORD = test
```

如果然后进行查询，例如

```
SELECT * FROM test_db.test_table
```

```
SELECT * FROM odbc('DSN=gregtest;Servername=some-server.com', 'test_db');
```

ODBC驱动程序将发送的值 `USERNAME` 和 `PASSWORD` 从 `odbc.ini` 到 `some-server.com`.

连接Postgresql的示例

Ubuntu操作系统。

为PostgreSQL安装unixODBC和ODBC驱动程序：

```
$ sudo apt-get install -y unixodbc odbcinst odbc-postgresql
```

配置 `/etc/odbc.ini` (或 `~/.odbc.ini`):

```
[DEFAULT]
Driver = myconnection

[myconnection]
Description      = PostgreSQL connection to my_db
Driver          = PostgreSQL Unicode
Database        = my_db
Servername      = 127.0.0.1
UserName        = username
Password        = password
Port            = 5432
Protocol        = 9.3
ReadOnly         = No
RowVersioning   = No
ShowSystemTables = No
ConnSettings    =
```

ClickHouse中的字典配置：

```
<yandex>
<dictionary>
  <name>table_name</name>
  <source>
    <odbc>
      <!-- You can specify the following parameters in connection_string: -->
      <!-- DSN=myconnection;UID=username;PWD=password;HOST=127.0.0.1;PORT=5432;DATABASE=my_db -->
      <connection_string>DSN=myconnection</connection_string>
      <table>postgresql_table</table>
    </odbc>
  </source>
  <lifetime>
    <min>300</min>
    <max>360</max>
  </lifetime>
  <layout>
    <hashed/>
  </layout>
  <structure>
    <id>
      <name>id</name>
    </id>
    <attribute>
      <name>some_column</name>
      <type>UInt64</type>
      <null_value>0</null_value>
```

```
</attribute>
</structure>
</dictionary>
</yandex>
```

或

```
CREATE DICTIONARY table_name (
    id UInt64,
    some_column UInt64 DEFAULT 0
)
PRIMARY KEY id
SOURCE(ODBC(connection_string 'DSN=myconnection' table 'postgresql_table'))
LAYOUT(HASHED())
LIFETIME(MIN 300 MAX 360)
```

您可能需要编辑 `odbc.ini` 使用驱动程序指定库的完整路径 `DRIVER=/usr/local/lib/psqlodbcw.so`.

连接MS SQL Server的示例

Ubuntu操作系统。

安装驱动程序:

```
$ sudo apt-get install tdsodbc freetds-bin sqsh
```

配置驱动程序:

```
$ cat /etc/freetds/freetds.conf
...
[MSSQL]
host = 192.168.56.101
port = 1433
tds version = 7.0
client charset = UTF-8

$ cat /etc/odbcinst.ini
...
[FreeTDS]
Description = FreeTDS
Driver      = /usr/lib/x86_64-linux-gnu/odbc/libtdsodbc.so
Setup       = /usr/lib/x86_64-linux-gnu/odbc/libtdsS.so
FileUsage   = 1
UsageCount  = 5

$ cat ~/.odbc.ini
...
[MSSQL]
Description = FreeTDS
Driver      = FreeTDS
Servername  = MSSQL
Database   = test
UID        = test
PWD        = test
Port       = 1433
```

在ClickHouse中配置字典：

```
<yandex>
  <dictionary>
    <name>test</name>
    <source>
      <odbc>
        <table>dict</table>
        <connection_string>DSN=MSSQL;UID=test;PWD=test</connection_string>
      </odbc>
    </source>

    <lifetime>
      <min>300</min>
      <max>360</max>
    </lifetime>

    <layout>
      <flat />
    </layout>

    <structure>
      <id>
        <name>k</name>
      </id>
      <attribute>
        <name>s</name>
        <type>String</type>
        <null_value></null_value>
      </attribute>
    </structure>
  </dictionary>
</yandex>
```

或

```
CREATE DICTIONARY test (
  k UInt64,
  s String DEFAULT ''
)
PRIMARY KEY k
SOURCE(ODBC(table 'dict' connection_string 'DSN=MSSQL;UID=test;PWD=test'))
LAYOUT(FLAT())
LIFETIME(MIN 300 MAX 360)
```

DBMS

Mysql

设置示例：

```
<source>
  <mysql>
    <port>3306</port>
    <user>clickhouse</user>
    <password>qwerty</password>
    <replica>
      <host>example01-1</host>
```

```

<priority>1</priority>
</replica>
<replica>
  <host>example01-2</host>
  <priority>1</priority>
</replica>
<db>db_name</db>
<table>table_name</table>
<where>id=10</where>
<invalidate_query>SQL_QUERY</invalidate_query>
</mysql>
</source>

```

或

```

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'
))

```

设置字段:

- `port` – The port on the MySQL server. You can specify it for all replicas, or for each one individually (inside `<replica>`).
- `user` – Name of the MySQL user. You can specify it for all replicas, or for each one individually (inside `<replica>`).
- `password` – Password of the MySQL user. You can specify it for all replicas, or for each one individually (inside `<replica>`).
- `replica` – Section of replica configurations. There can be multiple sections.

- `replica/host` – The MySQL host.
- `replica/priority` – The replica priority. When attempting to connect, ClickHouse traverses the replicas in order of priority. The lower the number, the higher the priority.

- `db` – Name of the database.
- `table` – Name of the table.
- `where` – The selection criteria. The syntax for conditions is the same as for `WHERE` 例如，mysql中的子句，`id > 10 AND id < 20`. 可选参数。
- `invalidate_query` – Query for checking the dictionary status. Optional parameter. Read more in the section [更新字典](#).

MySQL可以通过套接字在本地主机上连接。要做到这一点，设置 `host` 和 `socket`.

设置示例:

```
<source>
```

```
<source>
<mysql>
<host>localhost</host>
<socket>/path/to/socket/file.sock</socket>
<user>clickhouse</user>
<password>qwerty</password>
<db>db_name</db>
<table>table_name</table>
<where>id=10</where>
<invalidate_query>SQL_QUERY</invalidate_query>
</mysql>
</source>
```

或

```
SOURCE(MYSQL(
host 'localhost'
socket '/path/to/socket/file.sock'
user 'clickhouse'
password 'qwerty'
db 'db_name'
table 'table_name'
where 'id=10'
invalidate_query 'SQL_QUERY'
))
```

ClickHouse

设置示例：

```
<source>
<clickhouse>
<host>example01-01-1</host>
<port>9000</port>
<user>default</user>
<password></password>
<db>default</db>
<table>ids</table>
<where>id=10</where>
</clickhouse>
</source>
```

或

```
SOURCE(CLICKHOUSE(
host 'example01-01-1'
port 9000
user 'default'
password ""
db 'default'
table 'ids'
where 'id=10'
))
```

设置字段：

- **host** – The ClickHouse host. If it is a local host, the query is processed without any network activity. To improve fault tolerance, you can create a **分布** 表并在后续配置中输入它。
- **port** – The port on the ClickHouse server

- `port` – The port on the ClickHouse server.
- `user` – Name of the ClickHouse user.
- `password` – Password of the ClickHouse user.
- `db` – Name of the database.
- `table` – Name of the table.
- `where` – The selection criteria. May be omitted.
- `invalidate_query` – Query for checking the dictionary status. Optional parameter. Read more in the section [更新字典](#).

Mongodb

设置示例:

```
<source>
<mongodb>
  <host>localhost</host>
  <port>27017</port>
  <user></user>
  <password></password>
  <db>test</db>
  <collection>dictionary_source</collection>
</mongodb>
</source>
```

或

```
SOURCE(MONGO(
  host 'localhost'
  port 27017
  user ""
  password ""
  db 'test'
  collection 'dictionary_source'
))
```

设置字段:

- `host` – The MongoDB host.
- `port` – The port on the MongoDB server.
- `user` – Name of the MongoDB user.
- `password` – Password of the MongoDB user.
- `db` – Name of the database.
- `collection` – Name of the collection.

Redis

设置示例:

```
<source>
<redis>
  <host>localhost</host>
  <port>6379</port>
  <storage_type>simple</storage_type>
  <db_index>0</db_index>
</redis>
</source>
```

或

```
SOURCE(REDIS(
    host 'localhost'
    port 6379
    storage_type 'simple'
    db_index 0
))
```

设置字段:

- `host` – The Redis host.
- `port` – The port on the Redis server.
- `storage_type` – The structure of internal Redis storage using for work with keys. `simple` 适用于简单源和散列单键源, `hash_map` 用于具有两个键的散列源。不支持具有复杂键的范围源和缓存源。可以省略, 默认值为 `simple`.
- `db_index` – The specific numeric index of Redis logical database. May be omitted, default value is 0.

字典键和字段

该 `<structure>` 子句描述可用于查询的字典键和字段。

XML描述:

```
<dictionary>
    <structure>
        <id>
            <name>Id</name>
        </id>

        <attribute>
            <!-- Attribute parameters -->
        </attribute>

        ...
    </structure>
</dictionary>
```

属性在元素中描述:

- `<id>` — 键列.
- `<attribute>` — 数据列. 可以有多个属性。

DDL查询:

```
CREATE DICTIONARY dict_name (
    Id UInt64,
    -- attributes
)
PRIMARY KEY Id
...
```

查询正文中描述了属性:

- `PRIMARY KEY` — 键列
- `AttrName AttrType` — 数据列. 可以有多个属性。

键

ClickHouse 支持以下类型的键:

- 数字键。 `UInt64`. 在定义 `<id>` 标记或使用 `PRIMARY KEY` 关键字。
- 复合密钥。 组不同类型的值。 在标签中定义 `<key>` 或 `PRIMARY KEY` 关键字。

XML 结构可以包含 `<id>` 或 `<key>`. DDL-查询必须包含单个 `PRIMARY KEY`.

警告

不能将键描述为属性。

数字键

类型: `UInt64`.

配置示例:

```
<id>
  <name>Id</name>
</id>
```

配置字段:

- `name` - The name of the column with keys.

对于 DDL-查询:

```
CREATE DICTIONARY (
  Id UInt64,
  ...
)
PRIMARY KEY Id
...
```

- `PRIMARY KEY` - The name of the column with keys.

复合密钥

键可以是一个 `tuple` 从任何类型的字段。该 **布局** 在这种情况下，必须是 `complex_key_hashed` 或 `complex_key_cache`.

提示

复合键可以由单个元素组成。例如，这使得可以使用字符串作为键。

键结构在元素中设置 `<key>`. 键字段的格式与字典的格式相同 **属性**. 示例:

```
<structure>
<key>
  <attribute>
    <name>field1</name>
    <type>String</type>
  </attribute>
  <attribute>
    <name>field2</name>
    <type>UInt32</type>
  </attribute>
</key>
</structure>
```

```
</attribute>
...
</key>
...
```

或

```
CREATE DICTIONARY (
    field1 String,
    field2 String
    ...
)
PRIMARY KEY field1, field2
...
```

对于查询 `dictGet*` 函数中，一个元组作为键传递。示例：`dictGetString('dict_name', 'attr_name', tuple('string for field1', num_for_field2))`。

属性

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
)
```

配置字段：

标签	产品描述	必填项
<code>name</code>	列名称。	是
<code>type</code>	ClickHouse数据类型。 ClickHouse尝试将字典中的值转换为指定的数据类型。例如，对于MySQL，该字段可能是 <code>TEXT</code> , <code>VARCHAR</code> ，或 <code>BLOB</code> 在MySQL源表中，但它可以上传为 <code>String</code> 在克里克豪斯 可为空 不支持。	是
<code>null_value</code>	非现有元素的默认值。 在示例中，它是一个空字符串。你不能使用 <code>NULL</code> 在这个领域。	是

<code>expression</code>	产品描述 ClickHouse对该值执行。 表达式可以是远程SQL数据库中的列名。因此，您可以使用它为远程列创建别名。	必填项。
	默认值：无表达式。	
<code>hierarchical</code>	如果 <code>true</code> ，该属性包含当前键的父键值。看 分层字典 。 默认值： <code>false</code> .	非也。
<code>injective</code>	标志，显示是否 <code>id -> attribute</code> 图像是 注射 。 如果 <code>true</code> ，ClickHouse可以自动放置后 <code>GROUP BY</code> 子句注入字典的请求。通常它显着减少了这种请求的数量。 默认值： <code>false</code> .	非也。
<code>is_object_id</code>	显示是否通过以下方式对MongoDB文档执行查询的标志 <code>ObjectId</code> 。 默认值： <code>false</code> .	非也。

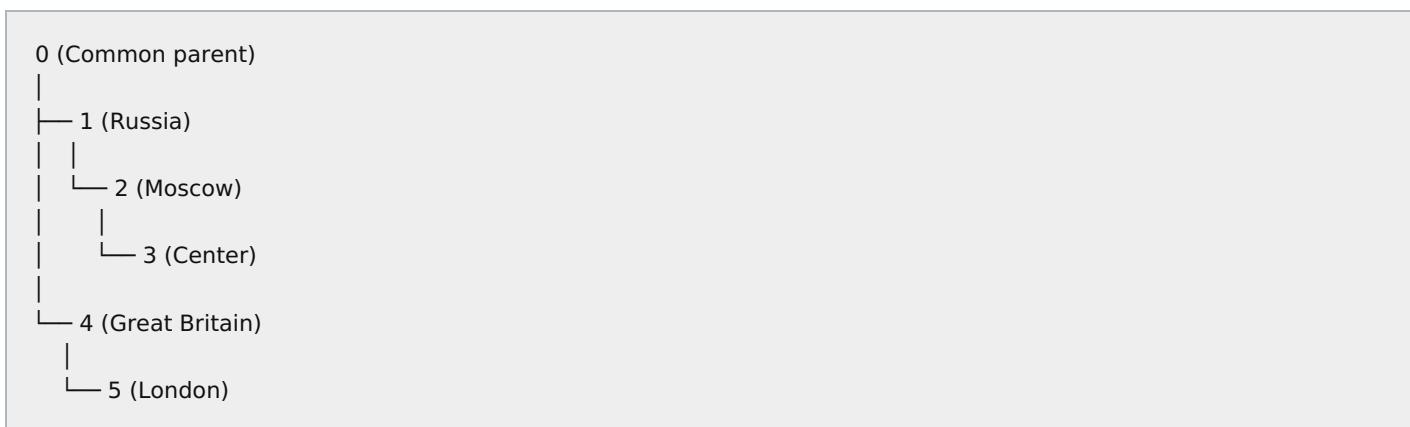
另请参阅

- [使用外部字典的函数](#).

分层字典

ClickHouse支持分层字典与 [数字键](#)。

看看下面的层次结构：



这种层次结构可以表示为下面的字典表。

<code>region_id</code>	<code>parent_region</code>	<code>region_name</code>
1	0	俄罗斯
2	1	莫斯科
3	2	中心
4	0	英国

5 region_id	4 parent_region	区域 region_name
----------------	--------------------	-------------------

此表包含一列 `parent_region` 包含该元素的最近父项的键。

ClickHouse 支持 **等级** 属性为 **外部字典** 属性。此属性允许您配置类似于上述的分层字典。

该 **独裁主义** 函数允许您获取元素的父链。

对于我们的例子，`dictionary` 的结构可以是以下内容：

```
<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>
```

内部字典

ClickHouse 包含用于处理地理数据库的内置功能。

这使您可以：

- 使用区域的 ID 以所需语言获取其名称。
- 使用区域 ID 获取城市、地区、联邦区、国家或大陆的 ID。
- 检查一个区域是否属于另一个区域。
- 获取父区域链。

所有功能支持 “translocality,” 能够同时使用不同的角度对区域所有权。有关详细信息，请参阅部分 “Functions for working with Yandex.Metrica dictionaries”.

在默认包中禁用内部字典。

要启用它们，请取消注释参数 `path_to_regions_hierarchy_file` 和 `path_to_regions_names_files` 在服务器配置文件中。

Geobase 从文本文件加载。

将 `regions_hierarchy*.txt` 文件到 `path_to_regions_hierarchy_file` 目录。此配置参数必须包含指向 `regions_hierarchy.txt` 文件（默认区域层次结构）和其他文件 (`regions_hierarchy_ua.txt`) 必须位于同一目录中。

把 `regions_names_*.txt` 在文件 `path_to_regions_names_files` 目录。

您也可以自己创建这些文件。文件格式如下：

`regions_hierarchy*.txt` : TabSeparated (无标题) , 列:

- 地区ID (UInt32)
- 父区域ID (UInt32)
- 区域类型 (UInt8) : 1-大陆, 3-国家, 4-联邦区, 5-地区, 6-城市; 其他类型没有价值
- 人口 (UInt32) — optional column

`regions_names_*.txt` : TabSeparated (无标题) , 列:

- 地区ID (UInt32)
- 地区名称 (String) — Can't contain tabs or line feeds, even escaped ones.

平面阵列用于存储在RAM中。出于这个原因, Id不应该超过一百万。

字典可以在不重新启动服务器的情况下更新。但是, 不会更新可用字典集。

对于更新, 将检查文件修改时间。如果文件已更改, 则更新字典。

检查更改的时间间隔在 `builtin_dictionaries_reload_interval` 参数。

字典更新(首次使用时加载除外)不会阻止查询。在更新期间, 查询使用旧版本的字典。如果在更新过程中发生错误, 则将错误写入服务器日志, 并使用旧版本的字典继续查询。

我们建议定期使用`geobase`更新字典。在更新期间, 生成新文件并将其写入单独的位置。一切准备就绪后, 将其重命名为服务器使用的文件。

还有与操作系统标识符和Yandex的工作功能。`Metrica`搜索引擎, 但他们不应该被使用。

Ansi Sql兼容性的ClickHouse SQL方言

注

本文依赖于表38, “Feature taxonomy and definition for mandatory features”, Annex F of ISO/IEC CD 9075-2:2013.

行为差异

下表列出了查询功能在ClickHouse中工作但行为不符合ANSI SQL中指定的情况。

Feature ID	功能名称	差异
E011	数字数据类型	带句点的数值文字被解释为近似值 (Float64) 而不是确切的 (Decimal)
E051-05	选择项目可以重命名	项目重命名具有比仅选择结果更广泛的可见性范围
E141-01	非空约束	<code>NOT NULL</code> 默认情况下, 表列隐含
E011-04	算术运算符	ClickHouse溢出而不是检查算法, 并根据自定义规则更改结果数据类型

功能状态

Feature ID	功能名称	状态	评论
E011	数字数据类型	部分	
E011-01	整型和小型数据类型	是	

E011-02 Feature ID	功能名称 类型数据类型	状态	FLOAT(<binary_precision>), REAL 和 DOUBLE PRECISION 不支持
E011-03	十进制和数值数据类型	部分	只有 DECIMAL(p,s) 支持，而不是 NUMERIC
E011-04	算术运算符	是	
E011-05	数字比较	是	
E011-06	数字数据类型之间的隐式转换	非也。	ANSI SQL允许在数值类型之间进行任意隐式转换，而ClickHouse依赖于具有多个重载的函数而不是隐式转换
E021	字符串类型	部分	
E021-01	字符数据类型	非也。	
E021-02	字符变化数据类型	非也。	String 行为类似，但括号中没有长度限制
E021-03	字符文字	部分	不自动连接连续文字和字符集支持
E021-04	字符长度函数	部分	非也。 USING 条款
E021-05	OCTET_LENGTH函数	非也。	LENGTH 表现类似
E021-06	SUBSTRING	部分	不支持 SIMILAR 和 ESCAPE 条款，否 SUBSTRING_REGEX 备选案文
E021-07	字符串联	部分	非也。 COLLATE 条款
E021-08	上下功能	是	
E021-09	修剪功能	是	
E021-10	固定长度和可变长度字符串类型之间的隐式转换	非也。	ANSI SQL允许在字符串类型之间进行任意隐式转换，而ClickHouse依赖于具有多个重载的函数而不是隐式转换
E021-11	职位功能	部分	不支持 IN 和 USING 条款，否 POSITION_REGEX 备选案文
E021-12	字符比较	是	
E031	标识符	部分	
E031-01	分隔标识符	部分	Unicode文字支持有限
E031-02	小写标识符	是	
E031-03	尾部下划线	是	
E051	基本查询规范	部分	
E051-01	SELECT DISTINCT	是	

Feature ID	SELECT DISTINCT 功能名称	状态	评论
E051-02	GROUP BY子句	是	
E051-04	分组依据可以包含不在列 <select list>	是	
E051-05	选择项目可以重命名	是	
E051-06	有条款	是	
E051-07	合格*在选择列表中	是	
E051-08	FROM子句中的关联名称	是	
E051-09	重命名FROM子句中的列	非也。	
E061	基本谓词和搜索条件	部分	
E061-01	比较谓词	是	
E061-02	谓词之间	部分	非也。 SYMMETRIC 和 ASYMMETRIC 条款
E061-03	在具有值列表的谓词中	是	
E061-04	像谓词	是	
E061-05	LIKE谓词：逃避条款	非也。	
E061-06	空谓词	是	
E061-07	量化比较谓词	非也。	
E061-08	存在谓词	非也。	
E061-09	比较谓词中的子查询	是	
E061-11	谓词中的子查询	是	
E061-12	量化比较谓词中的子查询	非也。	
E061-13	相关子查询	非也。	
E061-14	搜索条件	是	
E071	基本查询表达式	部分	
E071-01	UNION DISTINCT table	非	

Feature ID	功能名称	状态	评论
E071-02	联合所有表达算符	是	
E071-03	除了不同的表达算符	非 也。	
E071-05	通过表达算符组合的列不必具有完全相同的数据类型	是	
E071-06	子查询中的表达算符	是	
E081	基本特权	部分	正在进行的工作
E091	设置函数	是	
E091-01	AVG	是	
E091-02	COUNT	是	
E091-03	MAX	是	
E091-04	MIN	是	
E091-05	SUM	是	
E091-06	全部量词	非 也。	
E091-07	不同的量词	部分	并非所有聚合函数都受支持
E101	基本数据操作	部分	
E101-01	插入语句	是	注：ClickHouse中的主键并不意味着 UNIQUE 约束
E101-03	搜索更新语句	非 也。	有一个 ALTER UPDATE 批量数据修改语句
E101-04	搜索的删除语句	非 也。	有一个 ALTER DELETE 批量数据删除声明
E111	单行SELECT语句	非 也。	
E121	基本光标支持	非 也。	
E121-01	DECLARE CURSOR	非 也。	
E121-02	按列排序不需要在选择列表中	非 也。	

Feature ID	功能名称	列的值表达式	状态 也。	评论
E121-04	公开声明		非 也。	
E121-06	定位更新语句		非 也。	
E121-07	定位删除语句		非 也。	
E121-08	关闭声明		非 也。	
E121-10	FETCH语句：隐式NEXT		非 也。	
E121-17	使用保持游标		非 也。	
E131	空值支持（空值代替值）	部分		一些限制适用
E141	基本完整性约束	部分		
E141-01	非空约束	是		注: NOT NULL 默认情况下, 表列隐含
E141-02	非空列的唯一约束		非 也。	
E141-03	主键约束		非 也。	
E141-04	对于引用删除操作和引用更新操作，具有默认无操作的基本外键约束		非 也。	
E141-06	检查约束	是		
E141-07	列默认值	是		
E141-08	在主键上推断为非NULL	是		
E141-10	可以按任何顺序指定外键中的名称		非 也。	
E151	交易支持		非 也。	
E151-01	提交语句		非 也。	
E151-02	回滚语句		非 ,	

Feature ID	功能名称	状态	评论
E152	基本设置事务语句	非也。	
E152-01	SET TRANSACTION语句：隔离级别 SERIALIZABLE子句	非也。	
E152-02	SET TRANSACTION语句：只读和读写子句	非也。	
E153	具有子查询的可更新查询	非也。	
E161	SQL注释使用前导双减	是	
E171	SQLSTATE支持	非也。	
E182	主机语言绑定	非也。	
F031	基本架构操作	部分	
F031-01	CREATE TABLE语句创建持久基表	部分	非也。 SYSTEM VERSIONING, ON COMMIT, GLOBAL, LOCAL, PRESERVE, DELETE, REF IS, WITH OPTIONS, UNDER, LIKE, PERIOD FOR 子句，不支持用户解析的数据类型
F031-02	创建视图语句	部分	非也。 RECURSIVE, CHECK, UNDER, WITH OPTIONS 子句，不支持用户解析的数据类型
F031-03	赠款声明	是	
F031-04	ALTER TABLE语句：ADD COLUMN子句	部分	不支持 GENERATED 条款和系统时间段
F031-13	DROP TABLE语句： RESTRICT子句	非也。	
F031-16	DROP VIEW语句： RESTRICT子句	非也。	
F031-19	REVOKE语句：RESTRICT子句	非也。	
F041	基本连接表	部分	
F041-01	Inner join (但不一定是INNER关键字)	是	
F041-02	内部关键字	是	
F041-03	LEFT OUTER JOIN	是	

Feature ID	功能名称	状态	评论
F041-04	RIGHT OUTER JOIN	是	
F041-05	可以嵌套外部连接	是	
F041-07	左侧或右侧外部联接中的内部表也可用于内部联接	是	
F041-08	支持所有比较运算符（而不仅仅是=）	非 也。	
F051	基本日期和时间	部分	
F051-01	日期数据类型（包括对日期文字的支持）	部分	没有文字
F051-02	时间数据类型（包括对时间文字的支持），秒小数精度至少为0	非 也。	
F051-03	时间戳数据类型（包括对时间戳文字的支持），小数秒精度至少为0和6	非 也。	DateTime64 时间提供了类似的功能
F051-04	日期、时间和时间戳数据类型的比较谓词	部分	只有一种数据类型可用
F051-05	Datetime类型和字符串类型之间的显式转换	是	
F051-06	CURRENT_DATE	非 也。	today() 是相似的
F051-07	LOCALTIME	非 也。	now() 是相似的
F051-08	LOCALTIMESTAMP	非 也。	
F081	联盟和视图除外	部分	
F131	分组操作	部分	
F131-01	WHERE、GROUP BY和 HAVING子句在具有分组视图的查询中受支持	是	
F131-02	具有分组视图的查询中支持的多个表	是	
F131-03	设置具有分组视图的查询中支持的函数	是	
F131-04	具有分组依据和具有子句	是	

Feature ID	功能名称	状态	评论
F131-05	单行选择具有GROUP BY 和具有子句和分组视图	非 也。	
F181	多模块支持	非 也。	
F201	投函数	是	
F221	显式默认值	非 也。	
F261	案例表达式	是	
F261-01	简单案例	是	
F261-02	检索案例	是	
F261-03	NULLIF	是	
F261-04	COALESCE	是	
F311	架构定义语句	部分	
F311-01	CREATE SCHEMA	非 也。	
F311-02	为持久基表创建表	是	
F311-03	CREATE VIEW	是	
F311-04	CREATE VIEW: WITH CHECK OPTION	非 也。	
F311-05	赠款声明	是	
F471	标量子查询值	是	
F481	扩展空谓词	是	
F812	基本标记	非 也。	
T321	基本的SQL调用例程	非 也。	
T321-01	无重载的用户定义函数	非 也。	
T321-02	无重载的用户定义存储过 程	非 也。	

T321-03 Feature ID	函数调用 功能名称	非 状态	评论
T321-04	电话声明	非 也。	
T321-05	退货声明	非 也。	
T631	在一个列表元素的谓词中	是	

内省功能

您可以使用本章中描述的函数来反省 **ELF** 和 **DWARF** 用于查询分析。

警告

这些功能很慢，可能会强加安全考虑。

对于内省功能的正确操作：

- 安装 `clickhouse-common-static-dbg` 包。
- 设置 `allow_introspection_functions` 设置为 1。

For security reasons introspection functions are disabled by default.

ClickHouse 将探查器报告保存到 `trace_log` 系统表。确保正确配置了表和探查器。

addressToLine

将 ClickHouse 服务器进程内的虚拟内存地址转换为 ClickHouse 源代码中的文件名和行号。

如果您使用官方的 ClickHouse 软件包，您需要安装 `clickhouse-common-static-dbg` 包。

语法

```
addressToLine(address_of_binary_instruction)
```

参数

- `address_of_binary_instruction` (`UInt64`) — Address of instruction in a running process.

返回值

- 源代码文件名和此文件中用冒号分隔的行号。

For example, `/build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199` , where `199` is a line number.

- 二进制文件的名称，如果函数找不到调试信息。
- 空字符串，如果地址无效。

类型: 字符串。

示例

启用内省功能:

```
SET allow_introspection_functions=1
```

从中选择第一个字符串 `trace_log` 系统表:

```
SELECT * FROM system.trace_log LIMIT 1 \G
```

Row 1:

```
event_date: 2019-11-19
event_time: 2019-11-19 18:57:23
revision: 54429
timer_type: Real
thread_number: 48
query_id: 421b6855-1858-45a5-8f37-f383409d6d72
trace:
[140658411141617,94784174532828,94784076370703,94784076372094,94784076361020,94784175007680,140658
411116251,140658403895439]
```

该 `trace` 字段包含采样时的堆栈跟踪。

获取单个地址的源代码文件名和行号:

```
SELECT addressToLine(94784076370703) \G
```

Row 1:

```
addressToLine(94784076370703): /build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199
```

将函数应用于整个堆栈跟踪:

```
SELECT
    arrayStringConcat(arrayMap(x -> addressToLine(x), trace), '\n') AS trace_source_code_lines
FROM system.trace_log
LIMIT 1
\G
```

该 `arrayMap` 功能允许处理的每个单独的元素 `trace` 阵列由 `addressToLine` 功能。这种处理的结果，你在看 `trace_source_code_lines` 列的输出。

Row 1:

```
trace_source_code_lines: /lib/x86_64-linux-gnu/libpthread-2.27.so
/usr/lib/debug/usr/bin/clickhouse
/build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199
/build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.h:155
/usr/include/c++/9/bits/atomic_base.h:551
/usr/lib/debug/usr/bin/clickhouse
```

```
/lib/x86_64-linux-gnu/libpthread-2.27.so  
/build/glibc-OTsEL5/glibc-2.27/misc../sysdeps/unix/sysv/linux/x86_64/clone.S:97
```

addressToSymbol

将ClickHouse服务器进程内的虚拟内存地址转换为ClickHouse对象文件中的符号。

语法

```
addressToSymbol(address_of_binary_instruction)
```

参数

- `address_of_binary_instruction` (`UInt64`) — Address of instruction in a running process.

返回值

- 来自ClickHouse对象文件的符号。
- 空字符串，如果地址无效。

类型: `字符串`.

示例

启用内省功能:

```
SET allow_introspection_functions=1
```

从中选择第一个字符串 `trace_log` 系统表:

```
SELECT * FROM system.trace_log LIMIT 1 \G
```

Row 1:

```
event_date: 2019-11-20
event_time: 2019-11-20 16:57:59
revision: 54429
timer_type: Real
thread_number: 48
query_id: 724028bf-f550-45aa-910d-2af6212b94ac
trace:
[94138803686098,94138815010911,94138815096522,94138815101224,94138815102091,94138814222988,94138806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796144,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]
```

该 `trace` 字段包含采样时的堆栈跟踪。

获取单个地址的符号:

```
SELECT addressToSymbol(94138803686098) \G
```

Row 1:

```
addressToSymbol(94138803686098):
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE19a
...[REDACTED]
```

```
ddBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
```

将函数应用于整个堆栈跟踪：

```
SELECT
    arrayStringConcat(arrayMap(x -> addressToSymbol(x), trace), '\n') AS trace_symbols
FROM system.trace_log
LIMIT 1
\G
```

该 `arrayMap` 功能允许处理的每个单独的元素 `trace` 阵列由 `addressToSymbol` 功能。这种处理的结果，你在看 `trace_symbols` 列的输出。

Row 1:

```
trace_symbols:
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE19a
ddBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
_ZNK2DB10Aggregator21executeWithoutKeyImplERPcmPNS0_28AggregateFunctionInstructionEPNS_5ArenaE
_ZN2DB10Aggregator14executeOnBlockERSt6vectorIN3COWINS_7IColumnEE13immutable_ptrIS3_EESaIS6_EEmRNS_22A
gggregatedDataVariantsERS1_IPKS3_SaISC_EERS1_ISE_SaISE_EERb
_ZN2DB10Aggregator14executeOnBlockERKNS_5BlockERNS_22AggregatedDataVariantsERSt6vectorIPKNS_7IColumnES
aIS9_EERS6_ISB_SaISB_EERb
_ZN2DB10Aggregator7executeERKSt10shared_ptrINS_17IBlockInputStreamEERNS_22AggregatedDataVariantsE
_ZN2DB27AggregatingBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB26ExpressionBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB26ExpressionBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB28AsynchronousBlockInputStream9calculateEv
_ZNSt17_Function_handlerIFvvEZN2DB28AsynchronousBlockInputStream4nextEvEUIvE_E9_M_invokeERKSt9_Any_data
_ZN14ThreadpoolImpl20ThreadFromGlobalPoolE6workerESt14_List_iteratorIS0_E
_ZZN20ThreadFromGlobalPoolC4IZN14ThreadpoolImplIS_E12scheduleImplIvEET_St8functionIFvvEEiSt8optionalImEEUlvE
1_JEEEOS4_DpOT0_ENKUlvE_c1Ev
_ZN14ThreadpoolImplSt6threadE6workerESt14_List_iteratorIS0_E
execute_native_thread_routine
start_thread
clone
```

demangle

转换一个符号，您可以使用 `addressToSymbol` 函数到C++函数名。

语法

```
demangle(symbol)
```

参数

- `symbol` (`字符串`) — Symbol from an object file.

返回值

- C++函数的名称。
- 如果符号无效，则为空字符串。

类型: `字符串`.

示例

启用内省功能:

```
SET allow_introspection_functions=1
```

从中选择第一个字符串 `trace_log` 系统表:

```
SELECT * FROM system.trace_log LIMIT 1 \G
```

Row 1:

```
event_date: 2019-11-20
event_time: 2019-11-20 16:57:59
revision: 54429
timer_type: Real
thread_number: 48
query_id: 724028bf-f550-45aa-910d-2af6212b94ac
trace:
[94138803686098,94138815010911,94138815096522,94138815101224,94138815102091,94138814222988,94138806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796144,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]
```

该 `trace` 字段包含采样时的堆栈跟踪。

获取单个地址的函数名称:

```
SELECT demangle(addressToSymbol(94138803686098)) \G
```

Row 1:

```
demangle(addressToSymbol(94138803686098)): DB::IAggregateFunctionHelper<DB::AggregateFunctionSum<unsigned long, unsigned long, DB::AggregateFunctionSumData<unsigned long> > >::addBatchSinglePlace(unsigned long, char*, DB::IColumn const**, DB::Arena*) const
```

将函数应用于整个堆栈跟踪:

```
SELECT
    arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n') AS trace_functions
FROM system.trace_log
LIMIT 1
\G
```

该 `arrayMap` 功能允许处理的每个单独的元素 `trace` 阵列由 `demangle` 功能。这种处理的结果，你在看 `trace_functions` 列的输出。

Row 1:

```
trace_functions: DB::IAggregateFunctionHelper<DB::AggregateFunctionSum<unsigned long, unsigned long, DB::AggregateFunctionSumData<unsigned long> > >::addBatchSinglePlace(unsigned long, char*, DB::IColumn const**, DB::Arena*) const
DB::Aggregator::executeWithoutKeyImpl(char*&, unsigned long, DB::Aggregator::AggregateFunctionInstruction*, DB::Arena*) const
```

```
DB::Aggregator::executeOnBlock(std::vector<COW<DB::IColumn>::immutable_ptr<DB::IColumn>,
std::allocator<COW<DB::IColumn>::immutable_ptr<DB::IColumn> >, unsigned long, DB::AggregatedDataVariants&,
std::vector<DB::IColumn const*, std::allocator<DB::IColumn const*> >&, std::vector<std::vector<DB::IColumn const*,
std::allocator<DB::IColumn const*> >, std::allocator<std::vector<DB::IColumn const*, std::allocator<DB::IColumn
const*> >> >&, bool&)
DB::Aggregator::executeOnBlock(DB::Block const&, DB::AggregatedDataVariants&, std::vector<DB::IColumn const*,
std::allocator<DB::IColumn const*> >&, std::vector<std::vector<DB::IColumn const*, std::allocator<DB::IColumn
const*> >, std::allocator<std::vector<DB::IColumn const*, std::allocator<DB::IColumn const*> >> >&, bool&)
DB::Aggregator::execute(std::shared_ptr<DB::IBlockInputStream> const&, DB::AggregatedDataVariants&)
DB::AggregatingBlockInputStream::readImpl()
DB::IBlockInputStream::read()
DB::ExpressionBlockInputStream::readImpl()
DB::IBlockInputStream::read()
DB::ExpressionBlockInputStream::readImpl()
DB::IBlockInputStream::read()
DB::AsynchronousBlockInputStream::calculate()
std::_Function_handler<void (), DB::AsynchronousBlockInputStream::next()::{lambda()#1}>::_M_invoke(std::_Any_data
const&)
ThreadPoolImpl<ThreadFromGlobalPool>::worker(std::_List_iterator<ThreadFromGlobalPool>)
ThreadFromGlobalPool::ThreadFromGlobalPool<ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>
(std::function<void ()>, int, std::optional<unsigned long>:{lambda()#3}>
(ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>(std::function<void ()>, int, std::optional<unsigned
long>:{lambda()#3} &&:{lambda()#1}::operator()() const
ThreadPoolImpl<std::thread>::worker(std::_List_iterator<std::thread>)
execute_native_thread_routine
start_thread
clone
```

GEO函数

大圆形距离

使用 [great-circle distance](#) 公式 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： $[-180^\circ, 180^\circ]$ 。
- `lat1Deg` — 第一个点的纬度，单位：度，范围： $[-90^\circ, 90^\circ]$ 。
- `lon2Deg` — 第二个点的经度，单位：度，范围： $[-180^\circ, 180^\circ]$ 。
- `lat2Deg` — 第二个点的纬度，单位：度，范围： $[-90^\circ, 90^\circ]$ 。

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。

示例

```
SELECT greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)
```

```
greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673) |  
14132374.194975413 |
```

尖尖的人

检查指定的点是否至少包含在指定的一个椭圆中。
下述中的坐标是几何图形在笛卡尔坐标系中的位置。

```
pointInEllipses(x, y, x0, y0, a0, b0, ..., xn, yn, an, bn)
```

输入参数

- `x, y` — 平面上某个点的坐标。
- `xi, yi` — 第`i`个椭圆的中心坐标。
- `ai, bi` — 以`x, y`坐标为单位的第`i`个椭圆的轴。

输入参数的个数必须是 `2+4·n`，其中 `n` 是椭圆的数量。

返回值

如果该点至少包含在一个椭圆中，则返回 `1`；否则，则返回 `0`。

示例

```
SELECT pointInEllipses(55.755831, 37.617673, 55.755831, 37.617673, 1.0, 2.0)
```

```
| pointInEllipses(55.755831, 37.617673, 55.755831, 37.617673, 1., 2.) |  
| 1 |
```

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。元组类型，包含坐标的两个数字。
- `[(a, b), (c, d) ...]` — 多边形的顶点。阵列类型。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`。

示例

```
SELECT pointInPolygon((3., 3.), [(6, 0), (8, 4), (5, 8), (0, 2)]) AS res
```

```
| res |  
| 1 |
```

geohashEncode

将经度和纬度编码为 geohash-string，请参阅 (<http://geohash.org/>, <https://en.wikipedia.org/wiki/Geohash>)。

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

输入值

- 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）。

示例

```
SELECT geohashEncode(-5.60302734375, 42.593994140625, 0) AS res
```

```
res  
| ezs42d000000 |
```

geohashDecode

将任何 geohash 编码的字符串解码为 经度和纬度。

输入值

- encoded string - geohash 编码的字符串。

返回值

- (longitude, latitude) - 经度和纬度的 Float64 值的 2 元组。

示例

```
SELECT geohashDecode('ezs42') AS res
```

```
res  
| (-5.60302734375,42.60498046875) |
```

geoToH3

计算指定的分辨率的 H3 索引 (lon, lat)。

```
geoToH3(lon, lat, resolution)
```

输入值

- lon — 经度。Float64 类型。

- `lat` — 纬度。 `Float64` 类型。
- `resolution` — 索引的分辨率。 取值范围为: `[0, 15]`。 `UInt8` 类型。

返回值

- H3中六边形的索引值。
- 发生异常时返回0。

`UInt64` 类型。

示例

```
SELECT geoToH3(37.79506683, 55.71290588, 15) as h3Index
```

```
h3Index  
644325524701193974
```

geohashesInBox

计算在指定精度下计算最小包含指定的经纬范围的最小图形的geohash数组。

输入值

- `longitude_min` - 最小经度。其值应在 $[-180^\circ, 180^\circ]$ 范围内
- `latitude_min` - 最小纬度。其值应在 $[-90^\circ, 90^\circ]$ 范围内
- `longitude_max` - 最大经度。其值应在 $[-180^\circ, 180^\circ]$ 范围内
- `latitude_max` - 最大纬度。其值应在 $[-90^\circ, 90^\circ]$ 范围内
- `precision` - geohash的精度。其值应在 `[1, 12]` 内的 `UInt8` 类型的数字

请注意，上述所有的坐标参数必须同为 `Float32` 或 `Float64` 中的一种类型。

返回值

- 包含指定范围内的指定精度的geohash字符串数组。注意，您不应该依赖返回数组中geohash的顺序。
- `[]` - 当传入的最小经纬度大于最大经纬度时将返回一个空数组。

请注意，如果生成的数组长度超过10000时，则函数将抛出异常。

示例

```
SELECT geohashesInBox(24.48, 40.56, 24.785, 40.81, 4) AS thasos
```

```
thasos  
['sx1q','sx1r','sx32','sx1w','sx1x','sx38'] |
```

Hash 函数

Hash函数可以用于将元素不可逆的伪随机打乱。

halfMD5

计算字符串的MD5。然后获取结果的前8个字节并将它们作为 `UInt64` (大端) 返回。

此函数相当低效 (500万个短字符串/秒/核心)。

如果您不需要一定使用MD5，请使用'sipHash64'函数。

MD5

计算字符串的MD5并将结果放入`FixedString(16)`中返回。

如果您只是需要一个128位的hash，同时不需要一定使用MD5，请使用'sipHash128'函数。

如果您要获得与`md5sum`程序相同的输出结果，请使用`lower(hex(MD5(s)))`。

sipHash64

计算字符串的SipHash。

接受`String`类型的参数，返回`UInt64`。

SipHash是一种加密哈希函数。它的处理性能至少比MD5快三倍。

有关详细信息，请参阅链接：<https://131002.net/siphash/>

sipHash128

计算字符串的SipHash。

接受`String`类型的参数，返回`FixedString(16)`。

与`sipHash64`函数的不同在于它的最终计算结果为128位。

cityHash64

计算任意数量字符串的CityHash64或使用特定实现的Hash函数计算任意数量其他类型的Hash。

对于字符串，使用CityHash算法。这是一个快速的非加密哈希函数，用于字符串。

对于其他类型的参数，使用特定实现的Hash函数，这是一种快速的非加密的散列函数。

如果传递了多个参数，则使用CityHash组合这些参数的Hash结果。

例如，您可以计算整个表的checksum，其结果取决于行的顺序：`SELECT sum(cityHash64(*)) FROM table`。

intHash32

为任何类型的整数计算32位的哈希。

这是相对高效的非加密Hash函数。

intHash64

从任何类型的整数计算64位哈希码。

它的工作速度比`intHash32`函数快。

SHA1

SHA224

SHA256

计算字符串的SHA-1，SHA-224或SHA-256，并将结果字节集返回为`FixedString(20)`，`FixedString(28)`或`FixedString(32)`。

该函数相当低效（SHA-1大约500万个短字符串/秒/核心，而SHA-224和SHA-256大约220万个短字符串/秒/核心）。

我们建议仅在必须使用这些Hash函数且无法更改的情况下使用这些函数。

即使在这些情况下，我们仍建议将函数采用在写入数据时使用预计算的方式将其计算完毕。而不是在`SELECT`中计算它们。

URLHash(url[,N])

一种快速的非加密哈希函数，用于规范化的从URL获得的字符串。

`URLHash(s)` - 从一个字符串计算一个哈希，如果结尾存在尾随符号`/`，`?`或`#`则忽略。

`URLHash(s, N)` - 计算URL层次结构中字符串到N级别的哈希值，如果末尾存在尾随符号`/`，`?`或`#`则忽略。

URL的层级与`URLHierarchy`中的层级相同。此函数被用于Yandex.Metrica。

farmHash64

计算字符串的FarmHash64。

接受一个`String`类型的参数。返回`UInt64`。

有关详细信息，请参阅链接：[FarmHash64](#)

javaHash

计算字符串的JavaHash。

接受一个String类型的参数。返回Int32。

有关更多信息，请参阅链接：[JavaHash](#)

hiveHash

计算字符串的HiveHash。

接受一个String类型的参数。返回Int32。

与[JavaHash](#)相同，但不会返回负数。

metroHash64

计算字符串的MetroHash。

接受一个String类型的参数。返回UInt64。

有关详细信息，请参阅链接：[MetroHash64](#)

jumpConsistentHash

计算UInt64的JumpConsistentHash。

接受UInt64类型的参数。返回Int32。

有关更多信息，请参见链接：[JumpConsistentHash](#)

murmurHash2_32,murmurHash2_64

计算字符串的MurmurHash2。

接受一个String类型的参数。返回UInt64或UInt32。

有关更多信息，请参阅链接：[MurmurHash2](#)

murmurHash3_32,murmurHash3_64,murmurHash3_128

计算字符串的MurmurHash3。

接受一个String类型的参数。返回UInt64或UInt32或FixedString(16)。

有关更多信息，请参阅链接：[MurmurHash3](#)

xxHash32,xxHash64

计算字符串的xxHash。

接受一个String类型的参数。返回UInt64或UInt32。

有关更多信息，请参见链接：[xxHash](#)

IN运算符相关函数

in,notIn,globalIn,globalNotIn

请参阅[IN 运算符](#)部分。

tuple(x, y, ...), operator (x, y, ...)

函数用于对多个列进行分组。

对于具有类型T1, T2, ...的列，它返回包含这些列的元组 (T1, T2, ...)。执行该函数没有任何成本。

元组通常用作IN运算符的中间参数值，或用于创建lambda函数的形参列表。元组不能写入表。

元组元素(元组,n), 运算符x.N

函数用于从元组中获取列。

'N'是列索引，从1开始。N必须是常量正整数常数，并且不大于元组的大小。

执行该函数没有任何成本。

IP函数

IPv4NumToString(num)

接受一个UInt32（大端）表示的IPv4的地址，返回相应IPv4的字符串表现形式，格式为A.B.C.D（以点分割的十进制数字）。

IPv4StringToNum(s)

与IPv4NumToString函数相反。如果IPv4地址格式无效，则返回0。

IPv4NumToStringClassC(num)

与IPv4NumToString类似，但使用xxx替换最后一个字节。

示例：

```
SELECT
    IPv4NumToStringClassC(ClientIP) AS k,
    count() AS c
FROM test.hits
GROUP BY k
ORDER BY c DESC
LIMIT 10
```

k	c
83.149.9.xxx	26238
217.118.81.xxx	26074
213.87.129.xxx	25481
83.149.8.xxx	24984
217.118.83.xxx	22797
78.25.120.xxx	22354
213.87.131.xxx	21285
78.25.121.xxx	20887
188.162.65.xxx	19694
83.149.48.xxx	17406

由于使用'xxx'是不规范的，因此将来可能会更改。我们建议您不要依赖此格式。

IPv6NumToString(x)

接受FixedString(16)类型的二进制格式的IPv6地址。以文本格式返回此地址的字符串。

IPv6映射的IPv4地址以::ffff:111.222.33。例如：

```
SELECT IPv6NumToString(toFixedString(unhex('2A0206B800000000000000000000000000000011'), 16)) AS addr
```

addr
2a02:6b8::11

```
SELECT
    IPv6NumToString(ClientIP6 AS k),
    count() AS c
FROM hits_all
WHERE EventDate = today() AND substring(ClientIP6, 1, 12) != unhex('000000000000000000FFFF')
```

```
GROUP BY k  
ORDER BY c DESC  
LIMIT 10
```

IPv6NumToString(ClientIP6)		c
2a02:2168:aaa:bbbb::2 24695		
2a02:2698:abcd:abcd:abcd:8888:5555 22408		
2a02:6b8:0:fff::ff 16389		
2a01:4f8:111:6666::2 16016		
2a02:2168:888:222::1 15896		
2a01:7e00::ffff:ffff:ffff:222 14774		
2a02:8109:eee:ee:eeee:eeee:eeee:eee 14443		
2a02:810b:8888:888:8888:8888:8888:8888 14345		
2a02:6b8:0:444:4444:4444:4444:4444 14279		
2a01:7e00::ffff:ffff:ffff:ffff 13880		

```
SELECT  
    IPv6NumToString(ClientIP6 AS k),  
    count() AS c  
FROM hits_all  
WHERE EventDate = today()  
GROUP BY k  
ORDER BY c DESC  
LIMIT 10
```

IPv6NumToString(ClientIP6)		c
::ffff:94.26.111.111 747440		
::ffff:37.143.222.4 529483		
::ffff:5.166.111.99 317707		
::ffff:46.38.11.77 263086		
::ffff:79.105.111.111 186611		
::ffff:93.92.111.88 176773		
::ffff:84.53.111.33 158709		
::ffff:217.118.11.22 154004		
::ffff:217.118.11.33 148449		
::ffff:217.118.11.44 148243		

IPv6StringToNum(s)

与IPv6NumToString的相反。如果IPv6地址格式无效，则返回空字节字符串。
十六进制可以是大写的或小写的。

IPv4ToIntPv6(x)

接受一个UInt32类型的IPv4地址，返回FixedString(16)类型的IPv6地址。例如：

```
SELECT IPv6NumToString(IPv4ToIntPv6(IPv4StringToNum('192.168.0.1'))) AS addr
```

addr
::ffff:192.168.0.1

IPv6StringToNum(IPv6 地址)

接受一个FixedString(16)类型的IPv6地址，返回一个String，这个String中包含了删除指定位之后的地址的文本格式。例如：

```
WITH
IPv6StringToNum('2001:0DB8:AC10:FE01:FEED:BABE:CAFE:F00D') AS ipv6,
IPv4ToInt(IPv4StringToNum('192.168.0.1')) AS ipv4
SELECT
cutIPv6(ipv6, 2, 0),
cutIPv6(ipv4, 0, 2)
```

```
cutIPv6(ipv6, 2, 0) cutIPv6(ipv4, 0, 2)
| 2001:db8:ac10:fe01:feed:babe:cafe:0 | ::ffff:192.168.0.0 |
```

IPv4CIDRToRange(IPv4 地址, CIDR)

接受一个IPv4地址以及一个UInt8类型的CIDR。返回包含子网最低范围以及最高范围的元组。

```
SELECT IPv4CIDRToRange(toIPv4('192.168.5.2'), 16)
```

```
IPv4CIDRToRange(toIPv4('192.168.5.2'), 16)
| ('192.168.0.0','192.168.255.255') |
```

IPv6CIDRToRange(IPv6 地址, CIDR)

接受一个IPv6地址以及一个UInt8类型的CIDR。返回包含子网最低范围以及最高范围的元组。

```
SELECT IPv6CIDRToRange(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32);
```

```
IPv6CIDRToRange(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32)
| ('2001:db8::','2001:db8:ffff:ffff:ffff:ffff:ffff') |
```

toIPv4(字符串)

IPv4StringToNum() 的别名，它采用字符串形式的IPv4地址并返回IPv4类型的值，该二进制值等于IPv4StringToNum()返回的值。

```
WITH
'171.225.130.45' as IPv4_string
SELECT
toTypeName(IPv4StringToNum(IPv4_string)),
toTypeName(toIPv4(IPv4_string))
```

```
toTypeName(IPv4StringToNum(IPv4_string)) toTypeName(toIPv4(IPv4_string))
| UInt32 | IPv4 |
```

```
WITH
```

```
'171.225.130.45' as IPv4_string
SELECT
    hex(IPv4StringToNum(IPv4_string)),
    hex(toIPv4(IPv4_string))
```

```
hex(IPv4StringToNum(IPv4_string))——hex(toIPv4(IPv4_string))
| ABE1822D | ABE1822D |
```

toIPv6(字符串)

`IPv6StringToNum()` 的别名，它采用字符串形式的IPv6地址并返回IPv6类型的值，该二进制值等于`IPv6StringToNum()`返回的值。

```
WITH
    '2001:438:ffff::407d:1bc1' as IPv6_string
SELECT
    toTypeName(IPv6StringToNum(IPv6_string)),
    toTypeName(toIPv6(IPv6_string))
```

```
toTypeName(IPv6StringToNum(IPv6_string))——toTypeName(toIPv6(IPv6_string))
| FixedString(16) | IPv6 |
```

```
WITH
    '2001:438:ffff::407d:1bc1' as IPv6_string
SELECT
    hex(IPv6StringToNum(IPv6_string)),
    hex(toIPv6(IPv6_string))
```

```
hex(IPv6StringToNum(IPv6_string))——hex(toIPv6(IPv6_string))
| 20010438FFFF000000000000407D1BC1 | 20010438FFFF000000000000407D1BC1 |
```

JSON函数

在Yandex.Metrica中，用户使用JSON作为访问参数。为了处理这些JSON，实现了一些函数。（尽管在大多数情况下，JSON是预先进行额外处理的，并将结果值放在单独的列中。）所有的这些函数都进行了尽可能的假设。以使函数能够尽快的完成工作。

我们对JSON格式做了如下假设：

1. 字段名称（函数的参数）必须使常量。
2. 字段名称必须使用规范的编码。例如：`visitParamHas('{"abc":"def"}', 'abc') = 1`，但是`visitParamHas('"\u0061\u0062\u0063":"def"}', 'abc') = 0`
3. 函数可以随意的在多层嵌套结构下查找字段。如果存在多个匹配字段，则返回第一个匹配字段。
4. JSON除字符串文本外不存在空格字符。

ツ环板(ヨツ嘉ツツ偲青visヤツ静ヤツ青サツ催ヤツ涉)

检查是否存在«name»名称的字段

访问`paramextractuint(参数，名称)`

将名为«name»的字段的值解析成UInt64。如果这是一个字符串字段，函数将尝试从字符串的开头解析一个数字。如果该字段不存在，或无法从它中解析到数字，则返回0。

visitParamExtractInt(参数，名称)

与visitParamExtractUInt相同，但返回Int64。

访问paramextractfloat(参数，名称)

与visitParamExtractUInt相同，但返回Float64。

ツ环板(ヨツ嘉ツツ偲青妥-ツ姪(不ツ督ヨツ産)

解析true/false值。其结果是UInt8类型的。

掳胫((禄晦鹿脯露胫鲁隆鹿((酶-11-16""[肺陆(,,,)

返回字段的值，包含空格符。

示例：

```
visitParamExtractRaw('{"abc":"\\n\\u0000"}', 'abc') = "\\n\\u0000"  
visitParamExtractRaw('{"abc":{"def":[1,2,3]}}', 'abc') = '{"def":[1,2,3]}'
```

visitParamExtractString(参数，名称)

使用双引号解析字符串。这个值没有进行转义。如果转义失败，它将返回一个空白字符串。

示例：

```
visitParamExtractString('{"abc":"\\n\\u0000"}', 'abc') = '\\n\\0'  
visitParamExtractString('{"abc":"\\u263a"}', 'abc') = '☺'  
visitParamExtractString('{"abc":"\\u263"}', 'abc') = ""  
visitParamExtractString('{"abc":"hello"}', 'abc') = "
```

目前不支持\uXXXX\uYYYY这些字符编码，这些编码不在基本多文种平面中（它们被转化为CESU-8而不是UTF-8）。

以下函数基于[simdjson](#)，专为更复杂的JSON解析要求而设计。但上述假设2仍然适用。

JSONHas(json[, indices_or_keys]...)

如果JSON中存在该值，则返回1。

如果该值不存在，则返回0。

示例：

```
select JSONHas('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = 1  
select JSONHas('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', 4) = 0
```

indices_or_keys可以是零个或多个参数的列表，每个参数可以是字符串或整数。

- String = 按成员名称访问JSON对象成员。
- 正整数 = 从头开始访问第n个成员/成员名称。
- 负整数 = 从末尾访问第n个成员/成员名称。

您可以使用整数来访问JSON数组和JSON对象。

例如：

```
select JSONExtractKey('{"a": "hello", "b": [-100, 200.0, 300]}', 1) = 'a'  
select JSONExtractKey('{"a": "hello", "b": [-100, 200.0, 300]}', 2) = 'b'  
select JSONExtractKey('{"a": "hello", "b": [-100, 200.0, 300]}', -1) = 'b'  
select JSONExtractKey('{"a": "hello", "b": [-100, 200.0, 300]}', -2) = 'a'  
select JSONExtractString('{"a": "hello", "b": [-100, 200.0, 300]}', 1) = 'hello'
```

JSONLength(json[, indices_or_keys]...)

返回JSON数组或JSON对象的长度。

如果该值不存在或类型错误，将返回0。

示例：

```
select JSONLength('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = 3  
select JSONLength('{"a": "hello", "b": [-100, 200.0, 300]}') = 2
```

JSONType(json[, indices_or_keys]...)

返回JSON值的类型。

如果该值不存在，将返回Null。

示例：

```
select JSONType('{"a": "hello", "b": [-100, 200.0, 300]}') = 'Object'  
select JSONType('{"a": "hello", "b": [-100, 200.0, 300]}', 'a') = 'String'  
select JSONType('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = 'Array'
```

JSONExtractUInt(json[, indices_or_keys]...)

JSONExtractInt(json[, indices_or_keys]...)

JSONExtractFloat(json[, indices_or_keys]...)

JSONExtractBool(json[, indices_or_keys]...)

解析JSON并提取值。这些函数类似于visitParam*函数。

如果该值不存在或类型错误，将返回0。

示例：

```
select JSONExtractInt('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', 1) = -100  
select JSONExtractFloat('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', 2) = 200.0  
select JSONExtractUInt('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', -1) = 300
```

JSONExtractString(json[, indices_or_keys]...)

解析JSON并提取字符串。此函数类似于visitParamExtractString函数。

如果该值不存在或类型错误，则返回空字符串。

该值未转义。如果unescaping失败，则返回一个空字符串。

示例：

```
select JSONExtractString('{"a": "hello", "b": [-100, 200.0, 300]}', 'a') = 'hello'  
select JSONExtractString('{"abc":"\n\u0000"}', 'abc') = '\n\0'  
select JSONExtractString('{"abc":"\u263a"}', 'abc') = '☺'  
select JSONExtractString('{"abc":"\u2633"}', 'abc') = ''  
select JSONExtractString('{"abc":"hello"}', 'abc') = ''
```

JSONExtract(json[, indices_or_keys...], Return_type)

解析JSON并提取给定ClickHouse数据类型的值。

这是以前的`JSONExtract<type>`函数的变体。这意味着`JSONExtract(..., 'String')`返回与`JSONExtractString()`返回完全相同。`JSONExtract(..., 'Float64')`返回于`JSONExtractFloat()`返回完全相同。

示例：

```
SELECT JSONExtract('{"a": "hello", "b": [-100, 200.0, 300]}', 'Tuple(String, Array(Float64))') = ('hello',[-100,200,300])  
SELECT JSONExtract('{"a": "hello", "b": [-100, 200.0, 300]}', 'Tuple(b Array(Float64), a String)') = ([-100,200,300],'hello')  
SELECT JSONExtract('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', 'Array(Nullable(Int8))') = [-100, NULL, NULL]  
SELECT JSONExtract('{"a": "hello", "b": [-100, 200.0, 300]}', 'b', 4, 'Nullable(Int64)') = NULL  
SELECT JSONExtract('{"passed": true}', 'passed', 'UInt8') = 1  
SELECT JSONExtract('{"day": "Thursday"}', 'day', 'Enum8(\\"Sunday\\") = 0, \\"Monday\\") = 1, \\"Tuesday\\") = 2, \\"Wednesday\\") = 3, \\"Thursday\\") = 4, \\"Friday\\") = 5, \\"Saturday\\") = 6)') = 'Thursday'  
SELECT JSONExtract('{"day": 5}', 'day', 'Enum8(\\"Sunday\\") = 0, \\"Monday\\") = 1, \\"Tuesday\\") = 2, \\"Wednesday\\") = 3, \\"Thursday\\") = 4, \\"Friday\\") = 5, \\"Saturday\\") = 6)') = 'Friday'
```

JSONExtractKeysAndValues(json[, indices_or_keys...], Value_type)

从JSON中解析键值对，其中值是给定的ClickHouse数据类型。

示例：

```
SELECT JSONExtractKeysAndValues('{"x": {"a": 5, "b": 7, "c": 11}}', 'x', 'Int8') = [(('a',5),('b',7),('c',11))];
```

JSONExtractRaw(json[, indices_or_keys...])

返回JSON的部分。

如果部件不存在或类型错误，将返回空字符串。

示例：

```
select JSONExtractRaw('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = '[-100, 200.0, 300]'
```

Nullable处理函数

isNull

检查参数是否为`NULL`。

```
isNull(x)
```

参数

- `x` — 一个非复合数据类型的值。

返回值

- 1 如果 `x` 为 `NULL`。
- 0 如果 `x` 不为 `NULL`。

示例

存在以下内容的表

x	y
1	NULL
2	3

对其进行查询

```
:) SELECT x FROM t_null WHERE isNull(y)
```

```
SELECT x  
FROM t_null  
WHERE isNull(y)
```

x
1

```
1 rows in set. Elapsed: 0.010 sec.
```

isNotNull

检查参数是否不为 `NULL`。

```
isNotNull(x)
```

参数:

- `x` — 一个非复合数据类型的值。

返回值

- 0 如果 `x` 为 `NULL`。
- 1 如果 `x` 不为 `NULL`。

示例

存在以下内容的表

x	y
1	NULL
2	3

对其进行查询

```
:) SELECT x FROM t_null WHERE isNotNull(y)
```

```
SELECT x
```

```
SELECT x
FROM t_null
WHERE isNotNull(y)
```

x
2

1 rows in set. Elapsed: 0.010 sec.

合并

检查从左到右是否传递了«NULL»参数并返回第一个非'NULL'参数。

```
coalesce(x,...)
```

参数:

- 任何数量的非复合类型的参数。所有参数必须与数据类型兼容。

返回值

- 第一个非'NULL`参数。
- 'NULL'，如果所有参数都是'NULL`。

示例

考虑可以指定多种联系客户的方式的联系人列表。

name	mail	phone	icq
client 1	NULL	123-45-67	123
client 2	NULL	NULL	NULL

mail和phone字段是String类型，但icq字段是UInt32，所以它需要转换为String。

从联系人列表中获取客户的第一个可用联系方式：

```
:) SELECT coalesce(mail, phone, CAST(icq,'Nullable(String)')) FROM aBook
```

```
SELECT coalesce(mail, phone, CAST(icq, 'Nullable(String)'))
FROM aBook
```

name	coalesce(mail, phone, CAST(icq, 'Nullable(String)'))
client 1	123-45-67
client 2	NULL

2 rows in set. Elapsed: 0.006 sec.

ifNull

如果第一个参数为«NULL»，则返回第二个参数的值。

```
ifNull(x,alt)
```

参数:

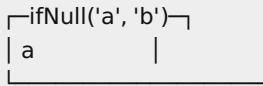
- `x` — 要检查«NULL»的值。
- `alt` — 如果`x`为'NULL`，函数返回的值。

返回值

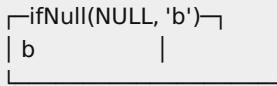
- 价值`x`，如果`x`不是`NULL`。
- 价值`alt`，如果`x`是`NULL`。

示例

```
SELECT ifNull('a', 'b')
```



```
SELECT ifNull(NULL, 'b')
```



nullIf

如果参数相等，则返回`NULL`。

```
nullIf(x, y)
```

参数:

`x, y` — 用于比较的值。它们必须是类型兼容的，否则将抛出异常。

返回值

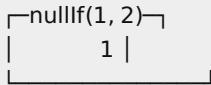
- 如果参数相等，则为`NULL`。
- 如果参数不相等，则为`x`值。

示例

```
SELECT nullIf(1, 1)
```



```
SELECT nullIf(1, 2)
```



assumeNotNull

将可为空类型的值转换为非`Nullable`类型的值。

```
assumeNotNull(x)
```

参数：

- `x` — 原始值。

返回值

- 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。

示例

存在如下 `t_null` 表。

```
SHOW CREATE TABLE t_null
```

statement

```
| CREATE TABLE default.t_null ( x Int8, y Nullable(Int8) ) ENGINE = TinyLog |
```

x	y
1	NULL
2	3

将列 `y` 作为 `assumeNotNull` 函数的参数。

```
SELECT assumeNotNull(y) FROM t_null
```

assumeNotNull(y)

```
| 0 |
| 3 |
```

```
SELECT toTypeName(assumeNotNull(y)) FROM t_null
```

toTypeName(assumeNotNull(y))

```
| Int8
| Int8 |
```

可调整

将参数的类型转换为 `Nullable`。

```
toNullable(x)
```

参数：

- `x` — 任何非复合类型的值。

返回值

- 输入的值，但其类型为 `Nullable`。

示例

```
SELECT toTypeName(10)
```

```
toNullable(10)
```

```

| toTypeName(10) |
| UInt8           |
|                 |

SELECT toTypeName(toNullable(10))

└─toTypeName(toNullable(10))─
| Nullable(UInt8)      |
|                         |

```

URL函数

所有这些功能都不遵循RFC。它们被最大程度简化以提高性能。

URL截取函数

如果URL中没有要截取的内容则返回空字符串。

协议

返回URL的协议。例如：http、ftp、mailto、magnet...

域

获取域名。

domainwithoutww

返回域名并删除第一个'www.'。

topLevelDomain

返回顶级域名。例如：.ru。

第一重要的元素分区域

返回«第一个有效子域名»。这并不是一个标准概念，仅用于Yandex.Metrica。如果顶级域名为'com'，'net'，'org'或者'co'则第一个有效子域名为二级域名。否则则返回三级域名。例如，`firstSignificantSubdomain('https://news.yandex.ru/')` = 'yandex'，`firstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex'。一些实现细节在未来可能会进行改变。

cutToFirstSignificantSubdomain

返回包含顶级域名与第一个有效子域名之间的内容（请参阅上面的内容）。

例如，`cutToFirstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex.com.tr'.

路径

返回URL路径。例如：`/top/news.html`，不包含请求参数。

pathFull

与上面相同，但包括请求参数和fragment。例如：`/top/news.html?page=2#comments`

查询字符串

返回请求参数。例如：`page=1&lr=213`。请求参数不包含问号已经# 以及# 之后所有的内容。

片段

返回URL的fragment标识。fragment不包含#。

querystring andfragment

返回请求参数和fragment标识。例如：`page=1#29390`。

extractURLParameter(URL,name)

返回URL请求参数中名称为'name'的参数。如果不存在则返回一个空字符串。如果存在多个匹配项则返回第一个相匹配的。此函数假设参数名称与参数值在url中的编码方式相同。

extractURLParameters(URL)

返回一个数组，其中以name=value的字符串形式返回url的所有请求参数。不以任何编码解析任何内容。

extractURLParameterNames(URL)

返回一个数组，其中包含url的所有请求参数的名称。不以任何编码解析任何内容。

URLHierarchy(URL)

返回一个数组，其中包含以/切割的URL的所有内容。?将被包含在URL路径以及请求参数中。连续的分割符号被记为一个。

Urlpathhierarchy(URL)

与上面相同，但结果不包含协议和host部分。/element(root)不包括在内。该函数用于在Yandex.Metric中实现导出URL的树形结构。

```
URLPathHierarchy('https://example.com/browse/CONV-6788') =  
[  
  '/browse',  
  '/browse/CONV-6788'  
]
```

decodeURLComponent(URL)

返回已经解码的URL。

例如：

```
SELECT decodeURLComponent('http://127.0.0.1:8123/?query=SELECT%201%3B') AS DecodedURL;
```

```
└─DecodedURL  
  ┌── http://127.0.0.1:8123/?query=SELECT 1; ──┘
```

删除URL中的部分内容

如果URL中不包含指定的部分，则URL不变。

cutWWW

删除开始的第一个'www.'。

cutQueryString

删除请求参数。问号也将被删除。

cutFragment

删除fragment标识。#同样也会被删除。

cutquerystring andfragment

删除请求参数以及fragment标识。问号以及#也会被删除。

cutURLParameter(URL,name)

删除URL中名称为'name'的参数。改函数假设参数名称以及参数值经过URL相同的编码。

UUID函数

下面列出了所有UUID的相关函数

generateUUIDv4

生成一个UUID（**版本4**）。

```
generateUUIDv4()
```

返回值

UUID类型的值。

使用示例

此示例演示如何在表中创建UUID类型的列，并对其写入数据。

```
:) CREATE TABLE t_uuid (x UUID) ENGINE=TinyLog  
:) INSERT INTO t_uuid SELECT generateUUIDv4()  
:) SELECT * FROM t_uuid
```

```
| f4bf890f-f9dc-4332-ad5c-0c18e73f28e9 |
```

toUUID(x)

将String类型的值转换为UUID类型的值。

```
toUUID(String)
```

返回值

UUID类型的值

使用示例

```
:) SELECT toUUID('61f0c404-5cb3-11e7-907b-a6006ad3dba0') AS uuid
```

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

UUIDStringToNum

接受一个String类型的值，其中包含36个字符且格式为xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx，将其转换为UUID的数值并以**固定字符串(16)**将其返回。

```
UUIDStringToNum(String)
```

返回值

固定字符串(16)

使用示例

```
:) SELECT
'612f3c40-5d3b-217e-707b-6a546a3d7b29' AS uuid,
UUIDStringToNum(uuid) AS bytes
```

```
uuid-----bytes-----
| 612f3c40-5d3b-217e-707b-6a546a3d7b29 | a/<@];!~p{jTj={} |
```

UUIDNumToString

接受一个**固定字符串(16)**类型的值，返回其对应的String表现形式。

```
UUIDNumToString(FixedString(16))
```

返回值

字符串。

使用示例

```
SELECT
'a/<@];!~p{jTj={} AS bytes,
UUIDNumToString(toFixedString(bytes, 16)) AS uuid
```

```
bytes-----uuid-----
| a/<@];!~p{jTj={} | 612f3c40-5d3b-217e-707b-6a546a3d7b29 |
```

另请参阅

- [dictgetuid](#)

arrayJoin 函数

这是一个非常有用的函数。

普通函数不会更改结果集的行数，而只是计算每行中的值（map）。

聚合函数将多行压缩到一行中（fold或reduce）。

'arrayJoin'函数获取每一行并将他们展开到多行（unfold）。

此函数将数组作为参数，并将该行在结果集中复制数组元素个数。

除了应用此函数的列中的值之外，简单地复制列中的所有值；它被替换为相应的数组值。

查询可以使用多个arrayJoin函数。在这种情况下，转换被执行多次。

请注意SELECT查询中的ARRAY JOIN语法，它提供了更广泛的可能性。

示例：

```
SELECT arrayJoin([1, 2, 3] AS src) AS dst, 'Hello', src
```

```
dst---\Hello\---src---
```

1	Hello	[1,2,3]
2	Hello	[1,2,3]
3	Hello	[1,2,3]

位图函数

位图函数用于对两个位图对象进行计算，对于任何一个位图函数，它都将返回一个位图对象，例如`and`, `or`, `xor`, `not`等。

位图对象有两种构造方法。一个是由聚合函数`groupBitmapState`构造的，另一个是由`Array Object`构造的。同时还可以将位图对象转化为数组对象。

我们使用`RoaringBitmap`实际存储位图对象，当基数小于或等于32时，它使用`Set`保存。当基数大于32时，它使用`RoaringBitmap`保存。这也是为什么低基数集的存储更快的原因。

有关`RoaringBitmap`的更多信息，请参阅：[呻吟声](#)。

bitmapBuild

从无符号整数数组构建位图对象。

```
bitmapBuild(array)
```

参数

- `array` – 无符号整数数组.

示例

```
SELECT bitmapBuild([1, 2, 3, 4, 5]) AS res
```

bitmapToArray

将位图转换为整数数组。

```
bitmapToArray(bitmap)
```

参数

- `bitmap` – 位图对象.

示例

```
SELECT bitmapToArray(bitmapBuild([1, 2, 3, 4, 5])) AS res
```

res
[1,2,3,4,5]

bitmapSubsetInRange

将位图指定范围（不包含`range_end`）转换为另一个位图。

```
bitmapSubsetInRange(bitmap, range_start, range_end)
```

参数

- `bitmap` – 位图对象.
- `range_start` – 范围起始点（含）.
- `range_end` – 范围结束点（不含）.

示例

```
SELECT
bitmapToArray(bitmapSubsetInRange(bitmapBuild([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,100,200,500]), toUInt32(30), toUInt32(200))) AS res
```

```
└── res
| [30,31,32,33,100] |
```

bitmapSubsetLimit

将位图指定范围（起始点和数目上限）转换为另一个位图。

```
bitmapSubsetLimit(bitmap, range_start, limit)
```

参数

- `bitmap` – 位图对象.
- `range_start` – 范围起始点（含）.
- `limit` – 子位图基数上限.

示例

```
SELECT
bitmapToArray(bitmapSubsetInRange(bitmapBuild([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,100,200,500]), toUInt32(30), toUInt32(200))) AS res
```

```
└── res
| [30,31,32,33,100,200,500] |
```

bitmapContains

检查位图是否包含指定元素。

```
bitmapContains(haystack, needle)
```

参数

- `haystack` – 位图对象.
- `needle` – 元素，类型 UInt32.

示例

```
SELECT bitmapContains(bitmapBuild([1,5,7,9]), toUInt32(9)) AS res
```

```
└─res  
  └─1
```

bitmapHasAny

与 `hasAny(array, array)` 类似，如果位图有任何公共元素则返回1，否则返回0。

对于空位图，返回0。

```
bitmapHasAny(bitmap,bitmap)
```

参数

- `bitmap` – bitmap 对象。

示例

```
SELECT bitmapHasAny(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
└─res  
  └─1
```

bitmapHasAll

与 `hasAll(array, array)` 类似，如果第一个位图包含第二个位图的所有元素，则返回1，否则返回0。

如果第二个参数是空位图，则返回1。

```
bitmapHasAll(bitmap,bitmap)
```

参数

- `bitmap` – bitmap 对象。

示例

```
SELECT bitmapHasAll(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
└─res  
  └─0
```

位图和

为两个位图对象进行与操作，返回一个新的位图对象。

```
bitmapAnd(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapToArray(bitmapAnd(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
res  
| [3] |
```

位图

为两个位图对象进行或操作，返回一个新的位图对象。

```
bitmapOr(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapToArray(bitmapOr(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
res  
| [1,2,3,4,5] |
```

bitmapXor

为两个位图对象进行异或操作，返回一个新的位图对象。

```
bitmapXor(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapToArray(bitmapXor(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
res  
| [1,2,4,5] |
```

bitmapAndnot

计算两个位图的差异，返回一个新的位图对象。

```
bitmapAndnot(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapToArray(bitmapAndnot(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
res  
| [1,2] |
```

bitmapCardinality

返回一个UInt64类型的数值，表示位图对象的基数。

```
bitmapCardinality(bitmap)
```

参数

- `bitmap` – 位图对象。

示例

```
SELECT bitmapCardinality(bitmapBuild([1, 2, 3, 4, 5])) AS res
```

```
res  
| 5 |
```

bitmapMin

返回一个UInt64类型的数值，表示位图中的最小值。如果位图为空则返回UINT32_MAX。

```
bitmapMin(bitmap)
```

参数

- `bitmap` – 位图对象。

示例

```
SELECT bitmapMin(bitmapBuild([1, 2, 3, 4, 5])) AS res
```

```
res  
| 1 |
```

bitmapMax

返回一个 UInt64 类型的数值，表示位图中的最大值。如果位图为空则返回 0。

```
bitmapMax(bitmap)
```

参数

- `bitmap` – 位图对象。

示例

```
SELECT bitmapMax(bitmapBuild([1, 2, 3, 4, 5])) AS res
```



位图和标准性

为两个位图对象进行与操作，返回结果位图的基数。

```
bitmapAndCardinality(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapAndCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```



bitmapOrCardinality

为两个位图进行或运算，返回结果位图的基数。

```
bitmapOrCardinality(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapOrCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
└─res  
  └─5
```

bitmapXorCardinality

为两个位图进行异或运算，返回结果位图的基数。

```
bitmapXorCardinality(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapXorCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
└─res  
  └─4
```

位图和非标准性

计算两个位图的差异，返回结果位图的基数。

```
bitmapAndnotCardinality(bitmap1,bitmap2)
```

参数

- `bitmap1` – 位图对象。
- `bitmap2` – 位图对象。

示例

```
SELECT bitmapAndnotCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
└─res  
  └─2
```

位操作函数

位操作函数适用于`UInt8`，`UInt16`，`UInt32`，`UInt64`，`Int8`，`Int16`，`Int32`，`Int64`，`Float32`或`Float64`中的任何类型。

结果类型是一个整数，其位数等于其参数的最大位。如果至少有一个参数为有符数字，则结果为有符数字。如果参数是浮点数，则将其强制转换为`Int64`。

bitAnd(a,b)

bitOr(a,b)

`bitXor(a,b)`
`bitNot(a)`
`bitShiftLeft(a,b)`
`bitShiftRight(a,b)`
`bitRotateLeft(a,b)`
`bitRotateRight(a,b)`
`bitTest(a,b)`
`bitTestAll(a,b)`
`bitTestAny(a,b)`

其他函数

主机名()

返回一个字符串，其中包含执行此函数的主机的名称。对于分布式处理，如果在远程服务器上执行此函数，则将返回远程服务器主机的名称。

basename

在最后一个斜杠或反斜杠后的字符串文本。此函数通常用于从路径中提取文件名。

```
basename( expr )
```

参数

- `expr` — 任何一个返回字符串结果的表达式。字符串

返回值

一个String类型的值，其包含：

- 在最后一个斜杠或反斜杠后的字符串文本内容。

如果输入的字符串以斜杆或反斜杆结尾，例如：`/` 或 `c:\`，函数将返回一个空字符串。

- 如果输入的字符串中不包含斜杆或反斜杠，函数返回输入字符串本身。

示例

```
SELECT 'some/long/path/to/file' AS a, basename(a)
```

```
a----- basename('some\\long\\path\\to\\file')-----  
| some\\long\\path\\to\\file | file |
```

```
SELECT 'some\\long\\path\\to\\file' AS a, basename(a)
```

```
a----- basename('some\\long\\path\\to\\file')  
| some\\long\\path\\to\\file | file
```

```
SELECT 'some-file-name' AS a, basename(a)
```

```
a----- basename('some-file-name')  
| some-file-name | some-file-name
```

visibleWidth(x)

以文本格式（以制表符分隔）向控制台输出值时，计算近似宽度。

系统使用此函数实现 Pretty 格式。

以文本格式（制表符分隔）将值输出到控制台时，计算近似宽度。

这个函数被系统用于实现漂亮的格式。

NULL 表示为对应于 NULL 在 Pretty 格式。

```
SELECT visibleWidth(NULL)
```

```
visibleWidth(NULL)  
4 |
```

toTypeName(x)

返回包含参数的类型名称的字符串。

如果将 NULL 作为参数传递给函数，那么它返回 Nullable (Nothing) 类型，它对应于 ClickHouse 中的内部 NULL。

块大小()

获取 Block 的大小。

在 ClickHouse 中，查询始终工作在 Block（包含列的部分的集合）上。此函数允许您获取调用其的块的大小。

实现(x)

将一个常量列变为一个非常量列。

在 ClickHouse 中，非常量列和常量列在内存中的表示方式不同。尽管函数对于常量列和非常量总是返回相同的结果，但它们的工作方式可能完全不同（执行不同的代码）。此函数用于调试这种行为。

ignore(...)

接受任何参数，包括 NULL。始终返回 0。

但是，函数的参数总是被计算的。该函数可以用于基准测试。

睡眠 (秒)

在每个 Block 上休眠 'seconds' 秒。可以是整数或浮点数。

sleepEachRow (秒)

在每行上休眠 'seconds' 秒。可以是整数或浮点数。

当前数据库()

返回当前数据库的名称。

当您需要在CREATE TABLE中的表引擎参数中指定数据库，您可以使用此函数。

isFinite(x)

接受Float32或Float64类型的参数，如果参数不是infinite且不是NaN，则返回1，否则返回0。

isInfinite(x)

接受Float32或Float64类型的参数，如果参数是infinite，则返回1，否则返回0。注意NaN返回0。

isNaN(x)

接受Float32或Float64类型的参数，如果参数是Nan，则返回1，否则返回0。

hasColumnInTable(['hostname'][, 'username'][, 'password']), 'database', 'table', 'column')

接受常量字符串：数据库名称、表名称和列名称。如果存在列，则返回等于1的UInt8常量表达式，否则返回0。如果设置了hostname参数，则测试将在远程服务器上运行。

如果表不存在，该函数将引发异常。

对于嵌套数据结构中的元素，该函数检查是否存在列。对于嵌套数据结构本身，函数返回0。

酒吧

使用unicode构建图表。

`bar(x, min, max, width)` 当 `x = max` 时，绘制一个宽度与 `(x - min)` 成正比且等于 `width` 的字符带。

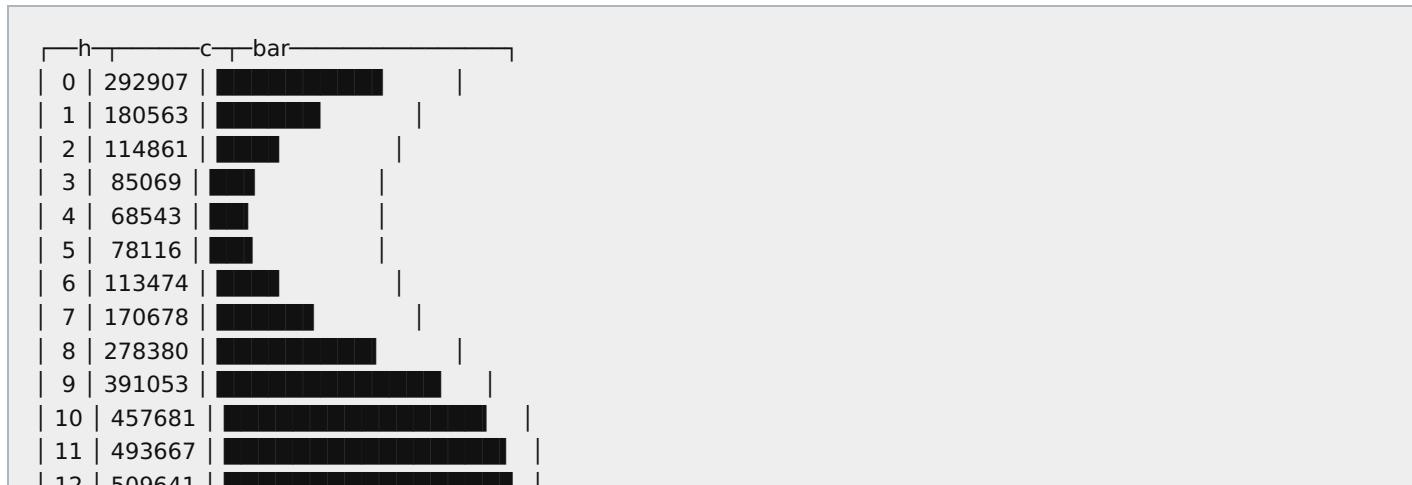
参数：

- `x` — 要显示的尺寸。
- `min, max` — 整数常量，该值必须是 Int64。
- `width` — 常量，可以是正整数或小数。

字符带的绘制精度是符号的八分之一。

示例：

```
SELECT
    toHour(EventTime) AS h,
    count() AS c,
    bar(c, 0, 600000, 20) AS bar
FROM test.hits
GROUP BY h
ORDER BY h ASC
```



13	522947
14	539954
15	528460
16	539201
17	523539
18	506467
19	520915
20	521665
21	542078
22	493642
23	400397

变换

根据定义，将某些元素转换为其他元素。

此函数有两种使用方式：

1. `transform(x, array_from, array_to, default)`

`x` – 要转换的值。

`array_from` – 用于转换的常量数组。

`array_to` – 将'from'中的值转换为的常量数组。

`default` – 如果'x'不等于'from'中的任何值，则默认转换的值。

`array_from` 和 `array_to` – 拥有相同大小的数组。

类型约束:

`transform(T, Array(T), Array(U), U) -> U`

`T`和`U`可以是`String`，`Date`，`DateTime`或任意数值类型的。

对于相同的字母（`T`或`U`），如果数值类型，那么它们不可不完全匹配的，只需要具备共同的类型即可。

例如，第一个参数是`Int64`类型，第二个参数是`Array(UInt16)`类型。

如果'x'值等于'array_from'数组中的一个元素，它将从'array_to'数组返回一个对应的元素（下标相同）。否则，它返回'default'。如果'array_from'匹配到了多个元素，则返回第一个匹配的元素。

示例:

```
SELECT
  transform(SearchEngineID, [2, 3], ['Yandex', 'Google'], 'Other') AS title,
  count() AS c
FROM test.hits
WHERE SearchEngineID != 0
GROUP BY title
ORDER BY c DESC
```

title	c
Yandex	498635
Google	229872
Other	104472

1. `transform(x, array_from, array_to)`

与第一种不同在于省略了'default'参数。

如果'x'值等于'array_from'数组中的一个元素，它将从'array_to'数组返回相应的元素（下标相同）。否则，它返回'x'。

类型约束：

```
transform(T, Array(T), Array(T)) -> T
```

示例：

```
SELECT
    transform(domain(Referer), ['yandex.ru', 'google.ru', 'vk.com'], ['www.yandex', 'example.com']) AS s,
    count() AS c
FROM test.hits
GROUP BY domain(Referer)
ORDER BY count() DESC
LIMIT 10
```

s	c
www.yandex	2906259
████████.ru	867767
████████.ru	313599
mail.yandex.ru	107147
████████.ru	100355
████████.ru	65040
news.yandex.ru	64515
████████.net	59141
example.com	57316

formatReadableSize(x)

接受大小（字节数）。返回带有后缀（KiB, MiB等）的字符串。

示例：

```
SELECT
    arrayJoin([1, 1024, 1024*1024, 192851925]) AS filesize_bytes,
    formatReadableSize(filesize_bytes) AS filesize
```

filesize_bytes	filesize
1	1.00 B
1024	1.00 KiB
1048576	1.00 MiB
192851925	183.92 MiB

至少(a,b)

返回a和b中的最小值。

最伟大(a,b)

返回a和b的最大值。

碌莽碌time拢time()

返回服务正常运行的秒数。

版本()

以字符串形式返回服务器的版本。

时区()

返回服务器的时区。

blockNumber

返回行所在的Block的序列号。

rowNumberInBlock

返回行所在Block中行的序列号。针对不同的Block始终重新计算。

rowNumberInAllBlocks()

返回行所在结果集中的序列号。此函数仅考虑受影响的Block。

运行差异(x)

计算数据块中相邻行的值之间的差异。

对于第一行返回0，并为每个后续行返回与前一行的差异。

函数的结果取决于受影响的Block和Block中的数据顺序。

如果使用ORDER BY创建子查询并从子查询外部调用该函数，则可以获得预期结果。

示例：

```
SELECT
    EventID,
    EventTime,
    runningDifference(EventTime) AS delta
FROM
(
    SELECT
        EventID,
        EventTime
    FROM events
    WHERE EventDate = '2016-11-24'
    ORDER BY EventTime ASC
    LIMIT 5
)
```

EventID	EventTime	delta
1106	2016-11-24 00:00:04	0
1107	2016-11-24 00:00:05	1
1108	2016-11-24 00:00:05	0
1109	2016-11-24 00:00:09	4
1110	2016-11-24 00:00:10	1

运行差异启动与第一值

与运行差异相同，区别在于第一行返回第一行的值，后续每个后续行返回与上一行的差值。

MACNumToString(num)

接受一个UInt64类型的数字。将其解释为big endian的MAC地址。返回包含相应MAC地址的字符串，格式为

AA:BB:CC:DD:EE:FF / 小写大写都可以

AA:BB:CC:DD:EE:FF (以冒号分隔的十六进制形式的数字)。

MACStringToNum(s)

与MACNumToString相反。如果MAC地址格式无效，则返回0。

MACStringToOUI(s)

接受格式为AA:BB:CC:DD:EE:FF (十六进制形式的冒号分隔数字) 的MAC地址。返回前三个八位字节作为UInt64编号。如果MAC地址格式无效，则返回0。

getSizeOfEnumType

返回枚举中的枚举数量。

```
getSizeOfEnumType(value)
```

参数:

- value — Enum 类型的值。

返回值

- Enum 的枚举数量。
- 如果类型不是Enum，则抛出异常。

示例

```
SELECT getSizeOfEnumType( CAST('a' AS Enum8('a' = 1, 'b' = 2) ) ) AS x
```

```
└── x  
  └── 2
```

toColumnName

返回在RAM中列的数据类型的名称。

```
toColumnName(value)
```

参数:

- value — 任何类型的值。

返回值

- 一个字符串，其内容是value在RAM中的类型名称。

toTypeName '与 ' **toColumnName** 的区别示例

```
:) select toTypeName(cast('2018-01-01 01:02:03' AS DateTime))
```

```
SELECT toTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))
```

```
└── toTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))  
  └── DateTime
```

```
1 rows in set. Elapsed: 0.008 sec.
```

```
:) select toColumnName(cast('2018-01-01 01:02:03' AS DateTime))
```

```
SELECT toColumnName(CAST('2018-01-01 01:02:03', 'DateTime'))
```

```
└─toColumnName(CAST('2018-01-01 01:02:03', 'DateTime'))─  
  |Const(UInt32)|
```

该示例显示 `DateTime` 数据类型作为 `Const(UInt32)` 存储在内存中。

dumpColumnStructure

输出在RAM中的数据结果的详细信息。

```
dumpColumnStructure(value)
```

参数:

- `value` — 任何类型的值。

返回值

- 一个字符串，其内容是 `value` 在 RAM 中的数据结构的详细描述。

示例

```
SELECT dumpColumnStructure(CAST('2018-01-01 01:02:03', 'DateTime'))
```

```
└─dumpColumnStructure(CAST('2018-01-01 01:02:03', 'DateTime'))─  
  |DateTime, Const(size = 1, UInt32(size = 1))|
```

defaultValueOfType

输出数据类型的默认值。

不包括用户设置的自定义列的默认值。

```
defaultValueOfType(expression)
```

参数:

- `expression` — 任意类型的值或导致任意类型值的表达式。

返回值

- 数值类型返回 0。
- 字符串类型返回空的字符串。
- 可为空类型返回 `NULL`。

示例

```
:) SELECT defaultValueOfType( CAST(1 AS Int8) )
```

```
SELECT defaultValueOfType(CAST(1, 'Int8'))
```

```
└─defaultValueOfType(CAST(1, 'Int8'))─
```

```
| 0 |  
1 rows in set. Elapsed: 0.002 sec.  
:) SELECT defaultValueOfArgumentType( CAST(1 AS Nullable(Int8) ) )  
  
SELECT defaultValueOfArgumentType(CAST(1, 'Nullable(Int8)' ))  
  
└defaultValueOfArgumentType(CAST(1, 'Nullable(Int8)'))─  
    ┌ NULL ─|  
    └─────────|  
1 rows in set. Elapsed: 0.002 sec.
```

复制

使用单个值填充一个数组。

用于[arrayJoin](#)的内部实现。

```
replicate(x, arr)
```

参数:

- `arr` — 原始数组。ClickHouse创建一个与原始数据长度相同的新数组，并用值`x`填充它。
- `x` — 生成的数组将被填充的值。

输出

- 一个被`x`填充的数组。

示例

```
SELECT replicate(1, ['a', 'b', 'c'])
```

```
└replicate(1, ['a', 'b', 'c'])─  
  ┌ [1,1,1] ─|  
  └─────────|
```

文件系统可用

返回磁盘的剩余空间信息（以字节为单位）。使用配置文件中的path配置评估此信息。

文件系统容量

返回磁盘的容量信息，以字节为单位。使用配置文件中的path配置评估此信息。

最后聚会

获取聚合函数的状态。返回聚合结果（最终状态）。

跑累积

获取聚合函数的状态并返回其具体的值。这是从第一行到当前行的所有行累计的结果。

例如，获取聚合函数的状态（示例`runningAccumulate(uniqState(UserID))`），对于数据块的每一行，返回所有先前行和当前行的状态合并后的聚合函数的结果。

因此，函数的结果取决于分区中数据块的顺序以及数据块中行的顺序。

`joinGet('join_storage_table_name', 'get_column', join_key)`

使用指定的连接键从Join类型引擎的表中获取数据。

`modelEvaluate(model_name, ...)`

使用外部模型计算。

接受模型的名称以及模型的参数。返回Float64类型的值。

`throwIf(x)`

如果参数不为零则抛出异常。

函数

ClickHouse中至少存在两种类型的函数 - 常规函数（它们称之为«函数»）和聚合函数。常规函数的工作就像分别为每一行执行一次函数计算一样（对于每一行，函数的结果不依赖于其他行）。聚合函数则从各行累积一组值（即函数的结果以来整个结果集）。

在本节中，我们将讨论常规函数。有关聚合函数，请参阅《聚合函数》一节。

* - 'arrayJoin'函数与表函数均属于第三种类型的函数。 *

强类型

与标准SQL相比，ClickHouse具有强类型。换句话说，它不会在类型之间进行隐式转换。每个函数适用于特定的一组类型。这意味着有时您需要使用类型转换函数。

常见的子表达式消除

查询中具有相同AST（相同语句或语法分析结果相同）的所有表达式都被视为具有相同的值。这样的表达式被连接并执行一次。通过这种方式也可以消除相同的子查询。

结果类型

所有函数都只能返回一个返回值。结果类型通常由参数的类型决定。但tupleElement函数（`a.N`运算符）和`toFixedString`函数是例外的。

常量

为了简单起见，某些函数的某些参数只能是常量。例如，`LIKE`运算符的右参数必须是常量。

几乎所有函数都为常量参数返回常量。除了用于生成随机数的函数。

'now'函数为在不同时间运行的查询返回不同的值，但结果被视为常量，因为常量在单个查询中很重要。

常量表达式也被视为常量（例如，`LIKE`运算符的右半部分可以由多个常量构造）。

对于常量和非常量参数，可以以不同方式实现函数（执行不同的代码）。但是，对于包含相同数据的常量和非常量参数它们的结果应该是一致的。

NULL值处理

函数具有以下行为：

- 如果函数的参数至少一个是«NULL»，则函数结果也是«NULL»。
- 在每个函数的描述中单独指定的特殊行为。在ClickHouse源代码中，这些函数具有«UseDefaultImplementationForNulls = false»。

不可变性

函数不能更改其参数的值 - 任何更改都将作为结果返回。因此，计算单独函数的结果不依赖于在查询中写入函数的顺序。

错误处理

如果数据无效，某些函数可能会抛出异常。在这种情况下，将取消查询并将错误信息返回给客户端。对于分布式处理，当其中一个服务器发生异常时，其他服务器也会尝试中止查询。

表达式参数的计算

在几乎所有编程语言中，某些函数可能无法预先计算其中一个参数。这通常是运算符`&&`，`||`和`?:`。

但是在ClickHouse中，函数（运算符）的参数总是被预先计算。这是因为一次评估列的整个部分，而不是分别计算每一行。

执行分布式查询处理的功能

对于分布式查询处理，在远程服务器上执行尽可能多的查询处理阶段，并且在请求者服务器上执行其余阶段（合并中间结果和之后的所有内容）。

这意味着可以在不同的服务器上执行功能。

例如，在查询`SELECT f (sum (g (x))) FROM distributed_table GROUP BY h (y)`中，

- 如果`distributed_table`至少有两个分片，则在远程服务器上执行函数`'g'`和`'h'`，并在请求服务器上执行函数`'f'`。
- 如果`distributed_table`只有一个分片，则在该分片的服务器上执行所有`'f'`，`'g'`和`'h'`功能。

函数的结果通常不依赖于它在哪个服务器上执行。但是，有时这很重要。

例如，使用字典的函数时将使用运行它们的服务器上存在的字典。

另一个例子是`hostName`函数，它返回运行它的服务器的名称，以便在`SELECT`查询中对服务器进行`GROUP BY`。

如果查询中的函数在请求服务器上执行，但您需要在远程服务器上执行它，则可以将其包装在`«any»`聚合函数中，或将其添加到`«GROUP BY»`中。

功能与Yandex的工作。梅特里卡词典

为了使下面的功能正常工作，服务器配置必须指定获取所有Yandex的路径和地址。梅特里卡字典。字典在任何这些函数的第一次调用时加载。如果无法加载引用列表，则会引发异常。

For information about creating reference lists, see the section «Dictionaries».

多个地理基

ClickHouse支持同时使用多个备选地理基（区域层次结构），以支持某些地区所属国家的各种观点。

该`'clickhouse-server'` config指定具有区域层次结构的文

件`:: <path_to_regions_hierarchy_file>/opt/geo/regions_hierarchy.txt</path_to_regions_hierarchy_file>`

除了这个文件，它还搜索附近有`_`符号和任何后缀附加到名称（文件扩展名之前）的文件。

例如，它还会找到该文件`/opt/geo/regions_hierarchy_ua.txt`，如果存在。

`ua`被称为字典键。对于没有后缀的字典，键是空字符串。

所有字典都在运行时重新加载（每隔一定数量的秒重新加载一次，如`builtin_dictionaries_reload_interval` config参数中定义，或默认情况下每小时一次）。但是，可用字典列表在服务器启动时定义一次。

All functions for working with regions have an optional argument at the end – the dictionary key. It is referred to as the geobase.

示例：

```
regionToCountry(RegionID) - Uses the default dictionary: /opt/geo/regions_hierarchy.txt
regionToCountry(RegionID, "") - Uses the default dictionary: /opt/geo/regions_hierarchy.txt
regionToCountry(RegionID, 'ua') - Uses the dictionary for the 'ua' key: /opt/geo/regions_hierarchy_ua.txt
```

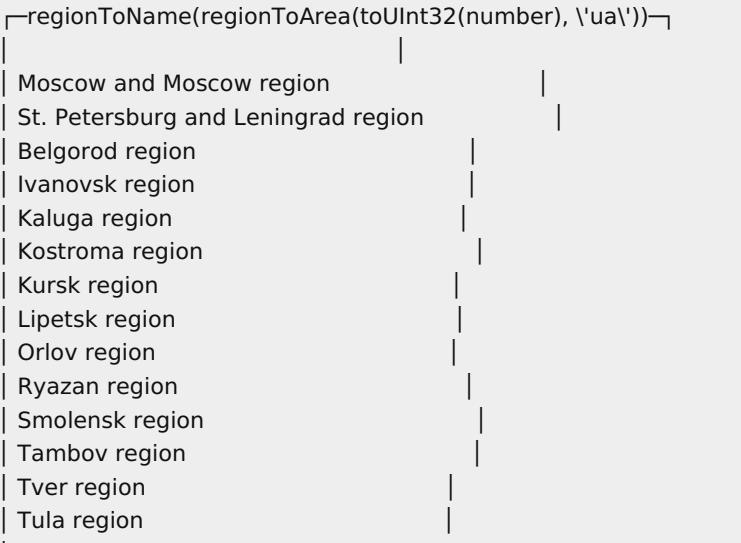
ツ環板(ヨツ嘉ツツ偲青regionシツ汎カツ鉄ツエツ渉])

Accepts a UInt32 number – the region ID from the Yandex geobase. If this region is a city or part of a city, it returns the region ID for the appropriate city. Otherwise, returns 0.

虏茅驴麓卤戮碌祿路戮魯拢])

将区域转换为区域（地理数据库中的类型5）。在所有其他方式，这个功能是一样的‘regionToCity’.

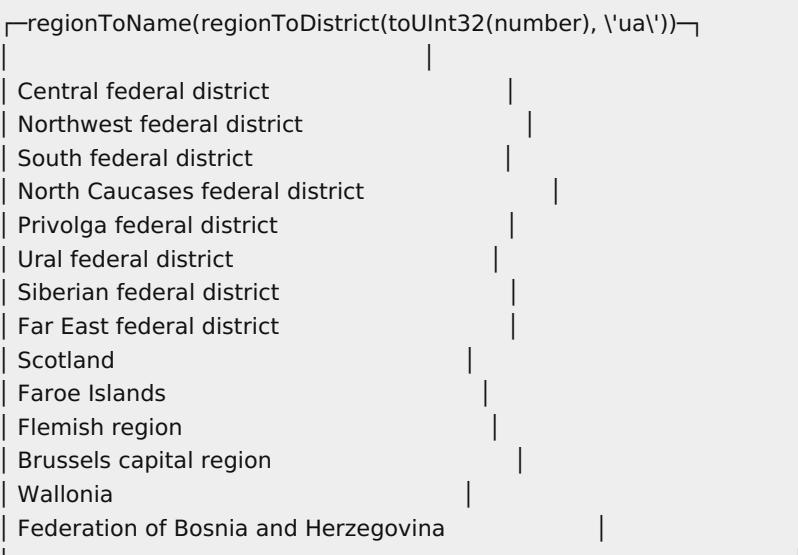
```
SELECT DISTINCT regionToName(regionToArea(toUInt32(number), 'ua'))  
FROM system.numbers  
LIMIT 15
```



regionToDistrict(id[,geobase])

将区域转换为联邦区（地理数据库中的类型4）。在所有其他方式，这个功能是一样的‘regionToCity’.

```
SELECT DISTINCT regionToName(regionToDistrict(toUInt32(number), 'ua'))  
FROM system.numbers  
LIMIT 15
```



虏茅驴麓卤戮碌祿路戮魯拢(陆毛隆隆(803)888-8325])

将区域转换为国家。在所有其他方式，这个功能是一样的‘regionToCity’.

示例: `regionToCountry(toUInt32(213)) = 225` 转换莫斯科 (213) 到俄罗斯 (225)。

據胫((碌脢鹿脢露胫譽隆鹿((酶-11-16""[肺陆,ase]))

将区域转换为大陆。在所有其他方式，这个功能是一样的‘regionToCity’。

示例: `regionToContinent(toUInt32(213)) = 10001` 将莫斯科 (213) 转换为欧亚大陆 (10001)。

ツ环板(ヨツ嘉ツッ偲青regionヤツ静ヤツ青サツ催ヤツ渉])

获取区域的人口。

The population can be recorded in files with the geobase. See the section «External dictionaries».

如果没有为该区域记录人口，则返回0。

在Yandex地理数据库中，可能会为子区域记录人口，但不会为父区域记录人口。

regionIn(lhs,rhs[,地理数据库])

检查是否‘lhs’属于一个区域‘rhs’区域。如果属于UInt8，则返回等于1的数字，如果不属于则返回0。

The relationship is reflexive – any region also belongs to itself.

ツ暗エツ氾环催ツ団ツ法ツ人])

Accepts a UInt32 number – the region ID from the Yandex geobase. Returns an array of region IDs consisting of the passed region and all parents along the chain.

示例: `regionHierarchy(toUInt32(213)) = [213,1,3,225,10001,10000]`.

地区名称(id[,郎])

Accepts a UInt32 number – the region ID from the Yandex geobase. A string with the name of the language can be passed as a second argument. Supported languages are: ru, en, ua, uk, by, kz, tr. If the second argument is omitted, the language ‘ru’ is used. If the language is not supported, an exception is thrown.

Returns a string – the name of the region in the corresponding language. If the region with the specified ID doesn't exist, an empty string is returned.

ua 和 uk 都意味着乌克兰。

取整函数

楼(x[,N])

返回小于或等于x的最大舍入数。该函数使用参数乘 $1/10^N$ ，如果 $1/10^N$ 不精确，则选择最接近的精确的适当数据类型的数。
‘N’是一个整数常量，可选参数。默认为0，这意味着不对其进行舍入。

‘N’可以是负数。

示例: `floor(123.45, 1) = 123.4, floor(123.45, -1) = 120.`

x 是任何数字类型。结果与其为相同类型。

对于整数参数，使用负‘N’值进行舍入是有意义的（对于非负‘N’，该函数不执行任何操作）。

如果取整导致溢出（例如，`floor(-128, -1)`），则返回特定于实现的结果。

ceil(x[,N]), 天花板(x[,N])

返回大于或等于‘x’的最小舍入数。在其他方面，它与‘floor’功能相同（见上文）。

圆形(x[,N])

将值取整到指定的小数位数。

该函数按顺序返回最近的数字。如果给定数字包含多个最近数字，则函数返回其中最接近偶数的数字（银行的取整方式）。

```
round(expression [, decimal_places])
```

参数：

- expression — 要进行取整的数字。可以是任何返回数字类型的表达式。

expression

- **decimal-places** — 整数类型。
 - 如果 **decimal-places > 0**，则该函数将值舍入小数点右侧。
 - 如果 **decimal-places < 0**，则该函数将小数点左侧的值四舍五入。
 - 如果 **decimal-places = 0**，则该函数将该值舍入为整数。在这种情况下，可以省略参数。

返回值：

与输入数字相同类型的取整后的数字。

示例

使用示例

```
SELECT number / 2 AS x, round(x) FROM system.numbers LIMIT 3
```

x	round(divide(number, 2))
0	0
0.5	0
1	1

取整的示例

取整到最近的数字。

```
round(3.2, 0) = 3  
round(4.1267, 2) = 4.13  
round(22,-1) = 20  
round(467,-2) = 500  
round(-467,-2) = -500
```

银行的取整。

```
round(3.5) = 4  
round(4.5) = 4  
round(3.55, 1) = 3.6  
round(3.65, 1) = 3.6
```

roundToExp2(num)

接受一个数字。如果数字小于1，则返回0。否则，它将数字向下舍入到最接近的（整个非负）2的x次幂。

圆形饱和度(num)

接受一个数字。如果数字小于1，则返回0。否则，它将数字向下舍入为集合中的数字：

1, 10, 30, 60, 120, 180, 240, 300, 600, 1200, 1800, 3600, 7200, 18000, 36000。此函数用于Yandex.Metrica报表中计算会话的持续时长。

圆数(num)

接受一个数字。如果数字小于18，则返回0。否则，它将数字向下舍入为集合中的数字：18, 25, 35, 45, 55。此函数用于Yandex.Metrica报表中用户年龄的计算。

roundDown(num,arr)

接受一个数字，将其向下舍入到指定数组中的元素。如果该值小于数组中的最低边界，则返回最低边界。

字典函数

有关连接和配置外部词典的信息，请参阅[外部词典](#)。

`dictGetUInt8,dictGetInt8,dictGetUInt16,dictGetInt16,dictGetUInt32,dictGetInt32,dictGetUInt64,dictGetInt64`

`dictGetFloat32,dictGetFloat64`

`dictGetDate,dictGetDateTime`

`dictGetUID`

`dictGetString`

```
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

```
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。

dictIsIn

```
dictIsIn ('dict_name', child_id, ancestor_id)
```

- 对于'`dict_name`'分层字典，查找'`child_id`'键是否位于'`ancestor_id`'内（或匹配'`ancestor_id`'）。返回`UInt8`。

独裁主义

```
dictGetHierarchy('dict_name', id)
```

- 对于'`dict_name`'分层字典，返回从'`id`'开始并沿父元素链继续的字典键数组。返回`Array (UInt64)`

dictHas

```
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

空

对于空字符串返回`1`，对于非空字符串返回`0`。

结果类型是`UInt8`。

如果字符串包含至少一个字节，则该字符串被视为非空字符串，即使这是一个空格或空字符。

该函数也适用于数组。

notEmpty

对于空字符串返回`0`，对于非空字符串返回`1`。

结果类型是`UInt8`。

该函数也适用于数组。

长度

返回字符串的字节长度。

结果类型是UInt64。

该函数也适用于数组。

长度8

假定字符串以UTF-8编码组成的文本，返回此字符串的Unicode字符长度。如果传入的字符串不是UTF-8编码，则函数可能返回一个预期外的值（不会抛出异常）。

结果类型是UInt64。

char_length,CHAR_LENGTH

假定字符串以UTF-8编码组成的文本，返回此字符串的Unicode字符长度。如果传入的字符串不是UTF-8编码，则函数可能返回一个预期外的值（不会抛出异常）。

结果类型是UInt64。

字符长度,字符长度

假定字符串以UTF-8编码组成的文本，返回此字符串的Unicode字符长度。如果传入的字符串不是UTF-8编码，则函数可能返回一个预期外的值（不会抛出异常）。

结果类型是UInt64。

低一点

将字符串中的ASCII转换为小写。

上,ucase

将字符串中的ASCII转换为大写。

lowerUTF8

将字符串转换为小写，函数假设字符串是以UTF-8编码文本的字符集。

同时函数不检测语言。因此对土耳其人来说，结果可能不完全正确。

如果UTF-8字节序列的长度对于代码点的大写和小写不同，则该代码点的结果可能不正确。

如果字符串包含一组非UTF-8的字节，则将引发未定义行为。

upperUTF8

将字符串转换为大写，函数假设字符串是以UTF-8编码文本的字符集。

同时函数不检测语言。因此对土耳其人来说，结果可能不完全正确。

如果UTF-8字节序列的长度对于代码点的大写和小写不同，则该代码点的结果可能不正确。

如果字符串包含一组非UTF-8的字节，则将引发未定义行为。

isValidUTF8

检查字符串是否为有效的UTF-8编码，是则返回1，否则返回0。

toValidUTF8

用◆ (U+FFFD) 字符替换无效的UTF-8字符。所有连续的无效字符都会被替换为一个替换字符。

```
toValidUTF8( input_string )
```

参数：

- `input_string` — 任何一个字符串类型的对象。

返回值：有效的UTF-8字符串。

示例

```
SELECT toValidUTF8('\x61\xF0\x80\x80\x80b')
```

```
└─toValidUTF8('a♦♦♦♦b')─  
| a♦b
```

反向

反转字符串。

reverseUTF8

以Unicode字符为单位反转UTF-8编码的字符串。如果字符串不是UTF-8编码，则可能获取到一个非预期的结果（不会抛出异常）。

format(pattern, s0, s1, ...)

使用常量字符串 pattern 格式化其他参数。pattern 字符串中包含由大括号 {} 包围的《替换字段》。未被包含在大括号中的任何内容都被视为文本内容，它将原样保留在返回值中。如果你需要在文本内容中包含一个大括号字符，它可以通过加倍来转义：{{ 和 {{' }}' }}。字段名称可以是数字（从零开始）或空（然后将它们视为连续数字）

```
SELECT format('{1} {0} {1}', 'World', 'Hello')
```

```
└─format('{1} {0} {1}', 'World', 'Hello')─  
| Hello World Hello
```

```
SELECT format('{}', 'Hello', 'World')
```

```
└─format('{}', 'Hello', 'World')─  
| Hello World
```

concat(s1, s2, ...)

将参数中的多个字符串拼接，不带分隔符。

concatAssumeInjective(s1, s2, ...)

与 concat 相同，区别在于，你需要保证 concat(s1, s2, s3) -> s4 是单射的，它将用于 GROUP BY 的优化。

子串

(s,offset,length),mid(s,offset,length),substr(s,offset,length)

以字节为单位截取指定位置字符串，返回以'offset'位置为开头，长度为'length'的子串。'offset'从1开始（与标准SQL相同）。'offset'和'length'参数必须是常量。

substringf8(s,offset,length)

与'substring'相同，但其操作单位为Unicode字符，函数假设字符串是以UTF-8进行编码的文本。如果不是则可能返回一个预期外的结果（不会抛出异常）。

appendTrailingCharIfAbsent(s,c)

如果's'字符串非空并且末尾不包含'c'字符，则将'c'字符附加到末尾。

convertCharset(s,from,to)

返回从'from'中的编码转换为'to'中的编码的字符串's'。

base64Encode(s)

将字符串's'编码成base64

base64Decode(s)

使用base64将字符串解码成原始字符串。如果失败则抛出异常。

tryBase64Decode(s)

使用base64将字符串解码成原始字符串。但如果出现错误，将返回空字符串。

endsWith(s,后缀)

返回是否以指定的后缀结尾。如果字符串以指定的后缀结束，则返回1，否则返回0。

开始使用 (s, 前缀)

返回是否以指定的前缀开头。如果字符串以指定的前缀开头，则返回1，否则返回0。

trimLeft(s)

返回一个字符串，用于删除左侧的空白字符。

trimRight(s)

返回一个字符串，用于删除右侧的空白字符。

trimBoth(s)

返回一个字符串，用于删除任一侧的空白字符。

字符串拆分合并函数

splitByChar (分隔符, s)

将字符串以'separator'拆分成多个子串。'separator'必须为仅包含一个字符的字符串常量。

返回拆分后的子串的数组。如果分隔符出现在字符串的开头或结尾，或者如果有多个连续的分隔符，则将在对应位置填充空的子串。

splitByString(分隔符, s)

与上面相同，但它使用多个字符的字符串作为分隔符。该字符串必须为非空。

arrayStringConcat(arr[,分隔符])

使用separator将数组中列出的字符串拼接起来。'separator'是一个可选参数：一个常量字符串，默认情况下设置为空字符串。

返回拼接后的字符串。

alphaTokens(s)

从范围a-z和A-Z中选择连续字节的子字符串。返回子字符串数组。

示例：

```
SELECT alphaTokens('abca1abc')
```

```
└─alphaTokens('abca1abc')─  
| ['abca','abc']      |
```

字符串搜索函数

下列所有函数在默认的情况下区分大小写。对于不区分大小写的搜索，存在单独的变体。

位置（大海捞针），定位（大海捞针）

在字符串 `haystack` 中搜索子串 `needle`。

返回子串的位置（以字节为单位），从1开始，如果未找到子串，则返回0。

对于不区分大小写的搜索，请使用函数 `positionCaseInsensitive`。

positionUTF8(大海捞针)

与 `position` 相同，但位置以 Unicode 字符返回。此函数工作在 UTF-8 编码的文本字符集中。如非此编码的字符集，则返回一些非预期结果（他不会抛出异常）。

对于不区分大小写的搜索，请使用函数 `positionCaseInsensitiveUTF8`。

多搜索分配（干草堆，[针₁，针₂，...，needle_n])

与 `position` 相同，但函数返回一个数组，其中包含所有匹配 `needle` 的位置。

对于不区分大小写的搜索或/和 UTF-8 格式，使用函

数 `multiSearchAllPositionsCaseInsensitive`，`multiSearchAllPositionsUTF8`，`multiSearchAllPositionsCaseInsensitiveUTF8`。

multiSearchFirstPosition(大海捞针,[针₁，针₂，...，needle_n])

与 `position` 相同，但返回在 `haystack` 中与 `needles` 字符串匹配的最左偏移。

对于不区分大小写的搜索或/和 UTF-8 格式，使用函

数 `multiSearchFirstPositionCaseInsensitive`，`multiSearchFirstPositionUTF8`，`multiSearchFirstPositionCaseInsensitiveUTF8`。

multiSearchFirstIndex(大海捞针,[针₁，针₂，...，needle_n])

返回在字符串 `haystack` 中最先查找到的 `needle` 的索引 `i`（从1开始），没有找到任何匹配项则返回0。

对于不区分大小写的搜索或/和 UTF-8 格式，使用函

数 `multiSearchFirstIndexCaseInsensitive`，`multiSearchFirstIndexUTF8`，`multiSearchFirstIndexCaseInsensitiveUTF8`。

多搜索（大海捞针，[针₁，针₂，...，needle_n])

如果 `haystack` 中至少存在一个 `needle` 匹配则返回1，否则返回0。

对于不区分大小写的搜索或/和 UTF-8 格式，使用函

数 `multiSearchAnyCaseInsensitive`，`multiSearchAnyUTF8`，`multiSearchAnyCaseInsensitiveUTF8`。

注意

在所有 `multiSearch*` 函数中，由于实现规范，`needles` 的数量应小于 2^8 。

匹配（大海捞针，模式）

检查字符串是否与 `pattern` 正则表达式匹配。`pattern` 可以是一个任意的 `re2` 正则表达式。`re2` 正则表达式的语法比 Perl 正则表达式的语法存在更多限制。

如果不匹配返回0，否则返回1。

请注意，反斜杠符号 (\) 用于在正则表达式中转义。由于字符串中采用相同的符号来进行转义。因此，为了在正则表达式中转义符号，必须在字符串文字中写入两个反斜杠 (\)。

正则表达式与字符串一起使用，就像它是一组字节一样。正则表达式中不能包含空字节。

对于在字符串中搜索子字符串的模式，最好使用LIKE或«position»，因为它们更加高效。

multiMatchAny (大海捞针, [模式₁, 模式₂, ..., pattern_n])

与match相同，但如果所有正则表达式都不匹配，则返回0；如果任何模式匹配，则返回1。它使用超扫描库。对于在字符串中搜索子字符串的模式，最好使用«multisearchany»，因为它更高效。

注意

任何haystack字符串的长度必须小于 2^{32} 字节，否则抛出异常。这种限制是因为hyperscan API而产生的。

multiMatchAnyIndex (大海捞针, [模式₁, 模式₂, ..., pattern_n])

与multiMatchAny相同，但返回与haystack匹配的任何内容的索引位置。

multiFuzzyMatchAny(干草堆, 距离, [模式₁, 模式₂, ..., pattern_n])

与multiMatchAny相同，但如果在haystack能够查找到任何模式匹配能够在指定的编辑距离内进行匹配，则返回1。此功能也处于实验模式，可能非常慢。有关更多信息，请参阅[hyperscan文档](#)。

multiFuzzyMatchAnyIndex(大海捞针, 距离, [模式₁, 模式₂, ..., pattern_n])

与multiFuzzyMatchAny相同，但返回匹配项的匹配能容的索引位置。

注意

multiFuzzyMatch*函数不支持UTF-8正则表达式，由于hyperscan限制，这些表达式被按字节解析。

注意

如要关闭所有hyperscan函数的使用，请设置SET allow_hyperscan = 0;。

提取 (大海捞针, 图案)

使用正则表达式截取字符串。如果'haystack'与'pattern'不匹配，则返回空字符串。如果正则表达式中不包含子模式，它将获取与整个正则表达式匹配的子串。否则，它将获取与第一个子模式匹配的子串。

extractAll (大海捞针, 图案)

使用正则表达式提取字符串的所有片段。如果'haystack'与'pattern'正则表达式不匹配，则返回一个空字符串。否则返回所有与正则表达式匹配的字符串数组。通常，行为与'extract'函数相同（它采用第一个子模式，如果没有子模式，则采用整个表达式）。

像 (干草堆, 模式) , 干草堆像模式运算符

检查字符串是否与简单正则表达式匹配。

正则表达式可以包含的元符号有%和_。

% 表示任何字节数（包括零字符）。

_ 表示任何一个字节。

可以使用反斜杠(\)来对元符号进行转义。请参阅«match»函数说明中有关转义的说明。

对于像`%needle%`这样的正则表达式，改函数与`position`函数一样快。

对于其他正则表达式，函数与`'match'`函数相同。

不喜欢（干草堆，模式），干草堆不喜欢模式运算符与`'like'`函数返回相反的结果。

大海捞针)

基于4-gram计算`haystack`和`needle`之间的距离：计算两个4-gram集合之间的对称差异，并用它们的基数和对其进行归一化。返回0到1之间的任何浮点数 - 越接近0则表示越多的字符串彼此相似。如果常量的`needle`或`haystack`超过32KB，函数将抛出异常。如果非常量的`haystack`或`needle`字符串超过32Kb，则距离始终为1。

对于不区分大小写的搜索或/和UTF-8格式，使用函

数`ngramDistanceCaseInsensitive`，`ngramDistanceUTF8`，`ngramDistanceCaseInsensitiveUTF8`。

ツ暗エツ汎环催ツ団ツ法ツ人)

与`ngramDistance`相同，但计算`needle`和`haystack`之间的非对称差异——`needle`的n-gram减去`needle`归一化n-gram。可用于模糊字符串搜索。

对于不区分大小写的搜索或/和UTF-8格式，使用函

数`ngramSearchCaseInsensitive`，`ngramSearchUTF8`，`ngramSearchCaseInsensitiveUTF8`。

注意

对于UTF-8，我们使用3-gram。所有这些都不是完全公平的n-gram距离。我们使用2字节哈希来散列n-gram，然后计算这些哈希表之间的（非）对称差异 - 可能会发生冲突。对于UTF-8不区分大小写的格式，我们不使用公平的`tolower`函数 - 我们将每个Unicode字符字节的第5位（从零开始）和字节的第一位归零 - 这适用于拉丁语，主要用于所有西里尔字母。

字符串替换函数

replaceOne (大海捞针，模式，更换)

用`'replacement'`子串替换`'haystack'`中与`'pattern'`子串第一个匹配的匹配项（如果存在）。

`'pattern'`和`'replacement'`必须是常量。

replaceAll (大海捞针，模式，替换)，替换 (大海捞针，模式，替 换)

用`'replacement'`子串替换`'haystack'`中出现的所有`'pattern'`子串。

replaceRegexpOne (大海捞针，模式，更换)

使用`'pattern'`正则表达式替换。`'pattern'`可以是任意一个有效的re2正则表达式。

如果存在与正则表达式匹配的匹配项，仅替换第一个匹配项。

同时`'replacement'`可以指定为正则表达式中的捕获组。可以包含`\1-\9`。

在这种情况下，函数将使用正则表达式的整个匹配项替换`\0`。使用其他与之对应的子模式替换对应的`\1-\9`。要在模版中使用“字符，请使用”将其转义。

另外还请记住，字符串文字需要额外的转义。

示例1. 将日期转换为美国格式：

```
SELECT DISTINCT
  EventDate,
  replaceRegexpOne(toString(EventDate), '(\d{4})-(\d{2})-(\d{2})', '$1/$2/$3') AS res
FROM test.hits
```

```
LIMIT 7  
FORMAT TabSeparated
```

2014-03-17	03/17/2014
2014-03-18	03/18/2014
2014-03-19	03/19/2014
2014-03-20	03/20/2014
2014-03-21	03/21/2014
2014-03-22	03/22/2014
2014-03-23	03/23/2014

示例2. 复制字符串十次：

```
SELECT replaceRegexpOne('Hello, World!', '.', '\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0') AS res
```

```
res
| Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello,
World!Hello, World! |
```

replaceRegexpAll (大海捞针，模式，替换)

与replaceRegexpOne相同，但会替换所有出现的匹配项。例如：

```
SELECT replaceRegexpAll('Hello, World!', '.', '\\0\\0') AS res
```

```
res
| HHeelllloo,, WWoorlldd!! |
```

例外的是，如果使用正则表达式捕获空白子串，则仅会进行一次替换。

示例：

```
SELECT replaceRegexpAll('Hello, World!', '^', 'here: ') AS res
```

```
res
| here: Hello, World! |
```

regexpQuoteMeta(s)

该函数用于在字符串中的某些预定义字符之前添加反斜杠。

预定义字符：'0', '\', '|', '(', ')', '^', '\$', '.', '[', ']', '?', '*', '+', '{', ':', '-'。

这个实现与re2 :: RE2 :: QuoteMeta略有不同。它以\0而不是00转义零字节，它只转义所需的字符。

有关详细信息，请参阅链接：[\[RE2\]](https://github.com/google/re2/blob/master/re2/re2.cc#L473) (<https://github.com/google/re2/blob/master/re2/re2.cc#L473>)

数学函数

以下所有的函数都返回一个Float64类型的数值。返回结果总是以尽可能最大精度返回，但还是可能与机器中可表示最接近该

值的数字不同。

e()

返回一个接近数学常量e的Float64数字。

pi()

返回一个接近数学常量π的Float64数字。

exp(x)

接受一个数值类型的参数并返回它的指数。

log(x),ln(x)

接受一个数值类型的参数并返回它的自然对数。

exp2(x)

接受一个数值类型的参数并返回它的2的x次幂。

log2(x)

接受一个数值类型的参数并返回它的底2对数。

exp10(x)

接受一个数值类型的参数并返回它的10的x次幂。

log10(x)

接受一个数值类型的参数并返回它的底10对数。

sqrt(x)

接受一个数值类型的参数并返回它的平方根。

cbrt(x)

接受一个数值类型的参数并返回它的立方根。

erf(x)

如果'x'是非负数，那么 `erf(x / σ√2)` 是具有正态分布且标准偏差为«σ»的随机变量的值与预期值之间的距离大于«x»。

示例（三西格玛准则）：

```
SELECT erf(3 / sqrt(2))
```

```
erf(divide(3, sqrt(2)))—  
| 0.9973002039367398 |
```

erfc(x)

接受一个数值参数并返回一个接近 $1 - \text{erf}(x)$ 的Float64数字，但不会丢失大«x»值的精度。

lgamma(x)

返回x的绝对值的自然对数的伽玛函数。

tgamma(x)

返回x的伽玛函数。

sin(x)

返回x的三角正弦值。

cos(x)

返回x的三角余弦值。

tan(x)

返回x的三角正切值。

asin(x)

返回x的反三角正弦值。

acos(x)

返回x的反三角余弦值。

atan(x)

返回x的反三角正切值。

pow(x,y),power(x,y)

接受x和y两个参数。返回x的y次方。

intExp2

接受一个数值类型的参数并返回它的2的x次幂 (UInt64)。

intExp10

接受一个数值类型的参数并返回它的10的x次幂 (UInt64)。

数组函数

空

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。

长度

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。

emptyArrayUInt8,emptyArrayUInt16,emptyArrayUInt32,emptyArrayInt8,emptyArrayInt16,emptyArrayInt32

`emptyArray(),emptyArrayInt(),emptyArrayInt2(),emptyArrayInt4()`

`emptyArrayFloat32,emptyArrayFloat64`

空空漫步，空空漫步时间

空字符串

不接受任何参数并返回适当类型的空数组。

`emptyArrayToSingle`

接受一个空数组并返回一个仅包含一个默认值元素的数组。

范围(N)

返回从0到N-1的数字数组。

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。

`array(x1, ...), operator [x1, ...]`

使用函数的参数作为数组元素创建一个数组。

参数必须是常量，并且具有最小公共类型的类型。必须至少传递一个参数，否则将不清楚要创建哪种类型的数组。也就是说，你不能使用这个函数来创建一个空数组（为此，使用上面描述的'emptyArray *'函数）。

返回'Array (T)'类型的结果，其中'T'是传递的参数中最小的公共类型。

`arrayConcat`

合并参数中传递的所有数组。

```
arrayConcat(arrays)
```

参数

- arrays – 任意数量的`阵列`类型的参数.

示例

```
SELECT arrayConcat([1, 2], [3, 4], [5, 6]) AS res
```

```
res  
| [1,2,3,4,5,6] |
```

`arrayElement(arr,n),运算符arr[n]`

从数组`arr`中获取索引为«n»的元素。`n`必须是任何整数类型。

数组中的索引从一开始。

支持负索引。在这种情况下，它选择从末尾开始编号的相应元素。例如，`arr [-1]`是数组中的最后一项。

如果索引超出数组的边界，则返回默认值（数字为0，字符串为空字符串等）。

`有(arr,elem)`

检查'arr'数组是否具有'elem'元素。

如果元素不在数组中，则返回0;如果在，则返回1。

`NULL` 值的处理。

```
SELECT has([1, 2, NULL, 1, NULL])
```

```
SELECT has([1, 2, NULL], NULL)
```

```
has([1, 2, NULL], NULL) └
```

```
  1 |
```

hasAll

检查一个数组是否是另一个数组的子集。

```
hasAll(set, subset)
```

参数

- `set` – 具有一组元素的任何类型的数组。
- `subset` – 任何类型的数组，其元素应该被测试为 `set` 的子集。

返回值

- `1`，如果 `set` 包含 `subset` 中的所有元素。
- `0`，否则。

特殊的定义

- 空数组是任何数组的子集。
- «Null»作为数组中的元素值进行处理。
- 忽略两个数组中的元素值的顺序。

示例

```
SELECT hasAll([], []) 返回1。
```

```
SELECT hasAll([1, Null], [Null]) 返回1。
```

```
SELECT hasAll([1.0, 2, 3, 4], [1, 3]) 返回1。
```

```
SELECT hasAll(['a', 'b'], ['a']) 返回1。
```

```
SELECT hasAll([1], ['a']) 返回0。
```

```
SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [3, 5]]) 返回0。
```

hasAny

检查两个数组是否存在交集。

```
hasAny(array1, array2)
```

参数

- `array1` – 具有一组元素的任何类型的数组。
- `array2` – 具有一组元素的任何类型的数组。

返回值

- `1`，如果 `array1` 和 `array2` 存在交集。
- `0`，否则。

特殊的定义

- «Null»作为数组中的元.素值进行处理。

- 忽略两个数组中的元素值的顺序。

卷之三

SELECT * FROM [M1_H2].[M1_H_1]; 返回 1

```
SELECT hasAny([ 128, 1, 512 ], [1]) 返回 1
```

```
SELECT hasAny([[1, 2], [3, 4]], ['a', 'c']) 返回 0
```

SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [1, 2]]) 返回 1.

`indexOf(arr,x)`

返回数组中第一个‘x’元素的索引（从1开始），如果‘x’元素不存在在数组中，则返回0。

示例：

:) SELECT indexOf([1,3,NULL,NULL],NULL)

```
SELECT indexOf([1, 3, NULL, NULL], NULL)
```

```
└─indexOf([1, 3, NULL, NULL], NULL)─  
|                                3 |
```

设置为«NULL»的元素将作为普通的元素值处理。

countEqual(arr,x)

返回数组中等于x的元素的个数。相当于arrayCount (elem -> elem = x, arr)。

NULL值将作为单独的元素值处理。

示例：

```
SELECT countEqual([1, 2, NULL, NULL], NULL)
```

└─countEqual([1, 2, NULL, NULL], NULL)┘

2 |

ツ暗エツ氾环催ツ団ツ法ツ人)

返回 Array [1, 2, 3, ..., length (arr)]

此功能通常与 **ARRAY JOIN** 一起使用。它允许在应用 **ARRAY JOIN** 后为每个数组计算一次。例如：

```
SELECT
    count() AS Reaches,
    countIf(num = 1) AS Hits
FROM test.hits
ARRAY JOIN
    GoalsReached,
    arrayEnumerate(GoalsReached) AS num
WHERE CounterID = 160656
LIMIT 10
```

Reaches	Hits
95606	31406

在此示例中，Reaches是转换次数（应用ARRAY JOIN后接收的字符串），Hits是浏览量（ARRAY JOIN之前的字符串）。在这种特殊情况下，您可以更轻松地获得相同的结果：

```
SELECT
    sum(length(GoalsReached)) AS Reaches,
    count() AS Hits
FROM test.hits
WHERE (CounterID = 160656) AND notEmpty(GoalsReached)
```

Reaches	Hits
95606	31406

此功能也可用于高阶函数。例如，您可以使用它来获取与条件匹配的元素的数组索引。

arrayEnumerateUniq(arr, ...)

返回与源数组大小相同的数组，其中每个元素表示与其下标对应的源数组元素在源数组中出现的次数。

例如：arrayEnumerateUniq ([10,20,10,30]) = [1,1,2,1]。

使用ARRAY JOIN和数组元素的聚合时，此函数很有用。

示例：

```
SELECT
    Goals.ID AS GoalID,
    sum(Sign) AS Reaches,
    sumIf(Sign, num = 1) AS Visits
FROM test.visits
ARRAY JOIN
    Goals,
    arrayEnumerateUniq(Goals.ID) AS num
WHERE CounterID = 160656
GROUP BY GoalID
ORDER BY Reaches DESC
LIMIT 10
```

GoalID	Reaches	Visits
53225	3214	1097
2825062	3188	1097
56600	2803	488
1989037	2401	365
2830064	2396	910
1113562	2372	373
3270895	2262	812
1084657	2262	345
56599	2260	799
3271094	2256	812

在此示例中，每个GoalID都计算转换次数（目标嵌套数据结构中的每个元素都是达到的目标，我们称之为转换）和会话数。

如果没有ARRAY JOIN，我们会将会话数计为总和（Sign）。但在这种特殊情况下，行乘以嵌套的Goals结构，因此为了在此之后计算每个会话一次，我们将一个条件应用于arrayEnumerateUniq (Goals.ID) 函数的值。

arrayEnumerateUniq函数可以使用与参数大小相同的多个数组。在这种情况下，对于所有阵列中相同位置的元素元组，考虑唯一性。

```
SELECT arrayEnumerateUniq([1, 1, 1, 2, 2, 2], [1, 1, 2, 1, 1, 2]) AS res
```

```
└── res  
  └── [1,2,1,1,2,1] |
```

当使用带有嵌套数据结构的ARRAY JOIN并在此结构中跨多个元素进一步聚合时，这是必需的。

arrayPopBack

从数组中删除最后一项。

```
arrayPopBack(array)
```

参数

- array – 数组。

示例

```
SELECT arrayPopBack([1, 2, 3]) AS res
```

```
└── res  
  └── [1,2] |
```

arrayPopFront

从数组中删除第一项。

```
arrayPopFront(array)
```

参数

- array – 数组。

示例

```
SELECT arrayPopFront([1, 2, 3]) AS res
```

```
└── res  
  └── [2,3] |
```

arrayPushBack

添加一个元素到数组的末尾。

```
arrayPushBack(array, single_value)
```

参数

- `array` – 数组。
- `single_value` – 单个值。只能将数字添加到带数字的数组中，并且只能将字符串添加到字符串数组中。添加数字时，ClickHouse会自动为数组的数据类型设置`single_value`类型。有关ClickHouse中数据类型的更多信息，请参阅[«数据类型»](#)。可以是'NULL'。该函数向数组添加一个«NULL»元素，数组元素的类型转换为`Nullable``。

示例

```
SELECT arrayPushBack(['a'], 'b') AS res
```

```
res  
| ['a','b'] |
```

arrayPushFront

将一个元素添加到数组的开头。

```
arrayPushFront(array, single_value)
```

参数

- `array` – 数组。
- `single_value` – 单个值。只能将数字添加到带数字的数组中，并且只能将字符串添加到字符串数组中。添加数字时，ClickHouse会自动为数组的数据类型设置`single_value`类型。有关ClickHouse中数据类型的更多信息，请参阅[«数据类型»](#)。可以是'NULL'。该函数向数组添加一个«NULL»元素，数组元素的类型转换为`Nullable``。

示例

```
SELECT arrayPushFront(['b'], 'a') AS res
```

```
res  
| ['a','b'] |
```

arrayResize

更改数组的长度。

```
arrayResize(array, size[, extender])
```

参数：

- `array` — 数组。
- `size` — 数组所需的长度。
 - 如果`size`小于数组的原始大小，则数组将从右侧截断。
- 如果`size`大于数组的初始大小，则使用`extender`值或数组项的数据类型的默认值将数组扩展到右侧。

- **extender** — 扩展数组的值。可以是'NULL`。

返回值:

一个 **size** 长度的数组。

调用示例

```
SELECT arrayResize([1], 3)
```

```
└─arrayResize([1], 3)─  
| [1,0,0] |
```

```
SELECT arrayResize([1], 3, NULL)
```

```
└─arrayResize([1], 3, NULL)─  
| [1,NULL,NULL] |
```

arraySlice

返回一个子数组，包含从指定位置的指定长度的元素。

```
arraySlice(array, offset[, length])
```

参数

- **array** – 数组。
- **offset** – 数组的偏移。正值表示左侧的偏移量，负值表示右侧的缩进值。数组下标从1开始。
- **length** - 子数组的长度。如果指定负值，则该函数返回 `[offset, array_length - length]`。如果省略该值，则该函数返回 `[offset, the_end_of_array]`。

示例

```
SELECT arraySlice([1, 2, NULL, 4, 5], 2, 3) AS res
```

```
└─res─  
| [2,NULL,4] |
```

设置为«NULL»的数组元素作为普通的数组元素值处理。

arraySort([func,] arr, ...)

以升序对 **arr** 数组的元素进行排序。如果指定了 **func** 函数，则排序顺序由 **func** 函数的调用结果决定。如果 **func** 接受多个参数，那么 **arraySort** 函数也将解析与 **func** 函数参数相同数量的数组参数。更详细的示例在 **arraySort** 的末尾。

整数排序示例:

```
SELECT arraySort([1, 3, 3, 0]);
```

```
└─arraySort([1, 3, 3, 0])─  
| [0,1,3,3] |
```

字符串排序示例：

```
SELECT arraySort(['hello', 'world', '!']);
```

```
arraySort(['hello', 'world', '!'])  
| ['!', 'hello', 'world'] |
```

NULL，NaN和Inf的排序顺序：

```
SELECT arraySort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf]);
```

```
arraySort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf])  
| [-inf, -4, 1, 2, 3, inf, nan, nan, NULL, NULL] |
```

- -Inf 是数组中的第一个。
- NULL 是数组中的最后一个。
- NaN 在 NULL 的前面。
- Inf 在 NaN 的前面。

注意：arraySort 是高阶函数。您可以将lambda函数作为第一个参数传递给它。在这种情况下，排序顺序由lambda函数的调用结果决定。

让我们来看一下如下示例：

```
SELECT arraySort((x) -> -x, [1, 2, 3]) as res;
```

```
res  
| [3, 2, 1] |
```

对于源数组的每个元素，lambda函数返回排序键，即[1 -> -1, 2 -> -2, 3 -> -3]。由于arraySort 函数按升序对键进行排序，因此结果为[3,2,1]。因此，(x) -> -x lambda函数将排序设置为降序。

lambda函数可以接受多个参数。在这种情况下，您需要为arraySort 传递与lambda参数个数相同的数组。函数使用第一个输入的数组中的元素组成返回结果；使用接下来传入的数组作为排序键。例如：

```
SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1]) as res;
```

```
res  
| ['world', 'hello'] |
```

这里，在第二个数组 ([2, 1]) 中定义了第一个数组 (['hello', 'world']) 的相应元素的排序键，即['hello' -> 2, 'world' -> 1]。由于lambda函数中没有使用x，因此源数组中的实际值不会影响结果的顺序。所以，'world'将是结果中的第一个元素，'hello'将是结果中的第二个元素。

其他示例如下所示。

```
SELECT arraySort((x, y) -> y, [0, 1, 2], ['c', 'b', 'a']) as res;
```

```
└── res  
  └── [2,1,0]
```

```
SELECT arraySort((x, y) -> -y, [0, 1, 2], [1, 2, 3]) as res;
```

```
└── res  
  └── [2,1,0]
```

注意

为了提高排序效率，使用了施瓦茨变换。

arrayReverseSort([func,] arr, ...)

以降序对 `arr` 数组的元素进行排序。如果指定了 `func` 函数，则排序顺序由 `func` 函数的调用结果决定。如果 `func` 接受多个参数，那么 `arrayReverseSort` 函数也将解析与 `func` 函数参数相同数量的数组作为参数。更详细的示例在 `arrayReverseSort` 的末尾。

整数排序示例：

```
SELECT arrayReverseSort([1, 3, 3, 0]);
```

```
└── arrayReverseSort([1, 3, 3, 0])  
  └── [3,3,1,0]
```

字符串排序示例：

```
SELECT arrayReverseSort(['hello', 'world', '!']);
```

```
└── arrayReverseSort(['hello', 'world', '!'])  
  └── ['world','hello','!']
```

`NULL`，`NaN` 和 `Inf` 的排序顺序：

```
SELECT arrayReverseSort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf]) as res;
```

```
└── res  
  └── [inf,3,2,1,-4,-inf,nan,nan,NULL,NULL]
```

- `Inf` 是数组中的第一个。

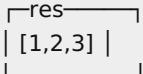
- `NULL` 是数组中的最后一个。

▼ **NULL** 数组中的取值

- **NaN** 在 **NULL** 的前面。
- **-Inf** 在 **NaN** 的前面。

注意：**arraySort** 是高阶函数。您可以将**lambda**函数作为第一个参数传递给它。如下示例所示。

```
SELECT arrayReverseSort((x) -> -x, [1, 2, 3]) AS res;
```



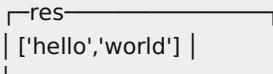
数组按以下方式排序：

数组按以下方式排序：

1. 首先，根据**lambda**函数的调用结果对源数组 ([1, 2, 3]) 进行排序。结果是[3, 2, 1]。
2. 反转上一步获得的数组。所以，最终的结果是[1, 2, 3]。

lambda函数可以接受多个参数。在这种情况下，您需要为 **arrayReverseSort** 传递与 **lambda**参数个数相同的数组。函数使用第一个输入的数组中的元素组成返回结果；使用接下来传入的数组作为排序键。例如：

```
SELECT arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) AS res;
```



在这个例子中，数组按以下方式排序：

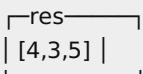
1. 首先，根据**lambda**函数的调用结果对源数组 (['hello', 'world']) 进行排序。其中，在第二个数组 ([2,1]) 中定义了源数组中相应元素的排序键。所以，排序结果['world', 'hello']。
2. 反转上一步骤中获得的排序数组。所以，最终的结果是['hello', 'world']。

其他示例如下所示。

```
SELECT arrayReverseSort((x, y) -> y, [4, 3, 5], ['a', 'b', 'c']) AS res;
```



```
SELECT arrayReverseSort((x, y) -> -y, [4, 3, 5], [1, 2, 3]) AS res;
```



arrayUniq(arr, ...)

如果传递一个参数，则计算数组中不同元素的数量。

如果传递了多个参数，则它计算多个数组中相应位置的不同元素元组的数量。

如果要获取数组中唯一的项的列表，可以使用arrayReduce ('groupUniqArray'，arr)。

arryjoin(arr)

一个特殊的功能。请参见[«ArrayJoin函数»](#)部分。

arrayDifference(arr)

返回一个数组，其中包含所有相邻元素对之间的差值。例如：

```
SELECT arrayDifference([1, 2, 3, 4])
```

```
arrayDifference([1, 2, 3, 4])
| [0,1,1,1] |
```

arrayDistinct(arr)

返回一个包含所有数组中不同元素的数组。例如：

```
SELECT arrayDistinct([1, 2, 2, 3, 1])
```

```
arrayDistinct([1, 2, 2, 3, 1])
| [1,2,3] |
```

arrayEnumerateDense(arr)

返回与源数组大小相同的数组，指示每个元素首次出现在源数组中的位置。例如：

arrayEnumerateDense ([10,20,10,30]) = [1,2,1,3]。

arrayIntersect(arr)

返回所有数组元素的交集。例如：

```
SELECT
arrayIntersect([1, 2], [1, 3], [2, 3]) AS no_intersect,
arrayIntersect([1, 2], [1, 3], [1, 4]) AS intersect
```

```
no_intersect intersect
| [ ] | [1] |
```

arrayReduce(agg_func, arr1, ...)

将聚合函数应用于数组并返回其结果。如果聚合函数具有多个参数，则此函数可应用于相同大小的多个数组。

arrayReduce ('agg_func'，arr1，...) - 将聚合函数agg_func应用于数组arr1...。如果传递了多个数组，则相应位置上的元素将作为多个参数传递给聚合函数。例如：SELECT arrayReduce ('max'，[1,2,3]) = 3

ツ暗エツ汎环催ツ団ツ法ツ人)

返回与源数组大小相同的数组，包含反转源数组的所有元素的结果。

时间日期函数

支持时区。

所有的时间日期函数都可以在第二个可选参数中接受时区参数。示例：Asia / Yekaterinburg。在这种情况下，它们使用指定的时区而不是本地（默认）时区。

```
SELECT
    toDateTime('2016-06-15 23:00:00') AS time,
    toDate(time) AS date_local,
    toDate(time, 'Asia/Yekaterinburg') AS date_yekat,
    toString(time, 'US/Samoa') AS time_samoa
```

time	date_local	date_yekat	time_samoa
2016-06-15 23:00:00	2016-06-15	2016-06-16	2016-06-15 09:00:00

仅支持与UTC相差一整小时的时区。

toTimeZone

将Date或DateTime转换为指定的时区。

玩一年

将Date或DateTime转换为包含年份编号（AD）的UInt16类型的数字。

到四分钟

将Date或DateTime转换为包含季度编号的UInt8类型的数字。

toMonth

将Date或DateTime转换为包含月份编号（1-12）的UInt8类型的数字。

今天一年

将Date或DateTime转换为包含一年中的某一天的编号的UInt16（1-366）类型的数字。

今天月

将Date或DateTime转换为包含一月中的某一天的编号的UInt8（1-31）类型的数字。

今天一周

将Date或DateTime转换为包含一周中的某一天的编号的UInt8（周一1，周日7）类型的数字。

toHour

将DateTime转换为包含24小时制（0-23）小时数的UInt8数字。

这个函数假设如果时钟向前移动，它是一个小时，发生在凌晨2点，如果时钟被移回，它是一个小时，发生在凌晨3点（这并非总是如此 - 即使在莫斯科时钟在不同的时间两次改变）。

toMinute

将DateTime转换为包含一小时中分钟数（0-59）的UInt8数字。

秒

将DateTime转换为包含一分钟中秒数（0-59）的UInt8数字。

闰秒不计算在内。

toUnixTimestamp

将DateTime转换为unix时间戳。

开始一年

将Date或DateTime向前取整到本年的第一天。

返回Date类型。

今年开始

将Date或DateTime向前取整到ISO本年的第一天。

返回Date类型。

四分之一开始

将Date或DateTime向前取整到本季度的第一天。

返回Date类型。

到月份开始

将Date或DateTime向前取整到本月的第一天。

返回Date类型。

注意

解析不正确日期的行为是特定于实现的。ClickHouse可能会返回零日期，抛出异常或执行«natural»溢出。

toMonday

将Date或DateTime向前取整到本周的星期一。

返回Date类型。

今天开始

将DateTime向前取整到当日的开始。

开始一小时

将DateTime向前取整到当前小时的开始。

to startofminute

将DateTime向前取整到当前分钟的开始。

to startoffiveminute

将DateTime以五分钟为单位向前取整到最接近的时间点。

开始分钟

将DateTime以十分钟为单位向前取整到最接近的时间点。

开始几分钟

将DateTime以十五分钟为单位向前取整到最接近的时间点。

toStartOfInterval(time_or_data, 间隔x单位[,time_zone])

这是名为 `toStartOf*` 的所有函数的通用函数。例如，

`toStartOfHour('2023-01-01 12:00:00')` 返回 `2023-01-01 00:00:00`，即日的午夜。

`toStartOfInterval (t, INTERVAL 1 year)` 返回与 `toStartOfYear (t)` 相同的结果，
`toStartOfInterval (t, INTERVAL 1 month)` 返回与 `toStartOfMonth (t)` 相同的结果，
`toStartOfInterval (t, INTERVAL 1 day)` 返回与 `toStartOfDay (t)` 相同的结果，
`toStartOfInterval (t, INTERVAL 15 minute)` 返回与 `toStartOfFifteenMinutes (t)` 相同的结果。

toTime

将 `DateTime` 中的日期转换为一个固定的日期，同时保留时间部分。

toRelativeYearNum

将 `Date` 或 `DateTime` 转换为年份的编号，从过去的某个固定时间点开始。

toRelativeQuarterNum

将 `Date` 或 `DateTime` 转换为季度的数字，从过去的某个固定时间点开始。

toRelativeMonthNum

将 `Date` 或 `DateTime` 转换为月份的编号，从过去的某个固定时间点开始。

toRelativeWeekNum

将 `Date` 或 `DateTime` 转换为星期数，从过去的某个固定时间点开始。

toRelativeDayNum

将 `Date` 或 `DateTime` 转换为当天的编号，从过去的某个固定时间点开始。

toRelativeHourNum

将 `DateTime` 转换为小时数，从过去的某个固定时间点开始。

toRelativeMinuteNum

将 `DateTime` 转换为分钟数，从过去的某个固定时间点开始。

toRelativeSecondNum

将 `DateTime` 转换为秒数，从过去的某个固定时间点开始。

toISOYear

将 `Date` 或 `DateTime` 转换为包含 ISO 年份的 `UInt16` 类型的编号。

toISOWeek

将 `Date` 或 `DateTime` 转换为包含 ISO 周数的 `UInt8` 类型的编号。

现在

不接受任何参数并在请求执行时的某一刻返回当前时间 (`DateTime`)。

此函数返回一个常量，即时请求需要很长能够完成。

今天

不接受任何参数并在请求执行时的某一刻返回当前日期 (`Date`)。

其功能与 '`toDate (now ())`' 相同。

昨天

不接受任何参数并在请求执行时的某一刻返回昨天的日期 (`Date`)。

其功能与 '`today () - 1`' 相同。

时隙

将时间向前取整半小时。

此功能用于Yandex.Metrica，因为如果跟踪标记显示单个用户的连续综合浏览量在时间上严格超过此数量，则半小时是将会话分成两个会话的最短时间。这意味着 (tag id, user id, time slot) 可用于搜索相应会话中包含的综合浏览量。

toyyymm

将Date或DateTime转换为包含年份和月份编号的UInt32类型的数字 (YYYY * 100 + MM)。

toyyymmdd

将Date或DateTime转换为包含年份和月份编号的UInt32类型的数字 (YYYY * 10000 + MM * 100 + DD)。

toYYYYMMDDhhmmss

将Date或DateTime转换为包含年份和月份编号的UInt64类型的数字 (YYYY * 10000000000 + MM * 100000000 + DD * 1000000 + hh * 10000 + mm * 100 + ss)。

隆隆隆路虏脢,,陇,貌,垄拢卢虏禄quar陇,貌路,隆拢脑枚脢虏,麓脢,脫,,,录,禄庐戮,utes,

函数将一段时间间隔添加到Date/DateTime，然后返回Date/DateTime。例如：

```
WITH
    toDate('2018-01-01') AS date,
    toDateTime('2018-01-01 00:00:00') AS date_time
SELECT
    addYears(date, 1) AS add_years_with_date,
    addYears(date_time, 1) AS add_years_with_date_time
```

add_years_with_date	add_years_with_date_time
2019-01-01	2019-01-01 00:00:00

subtractYears,subtractMonths,subtractWeeks,subtractDays,su

函数将Date/DateTime减去一段时间间隔，然后返回Date/DateTime。例如：

```
WITH
    toDate('2019-01-01') AS date,
    toDateTime('2019-01-01 00:00:00') AS date_time
SELECT
    subtractYears(date, 1) AS subtract_years_with_date,
    subtractYears(date_time, 1) AS subtract_years_with_date_time
```

subtract_years_with_date	subtract_years_with_date_time
2018-01-01	2018-01-01 00:00:00

dateDiff('unit',t1,t2,[时区])

返回以'unit'为单位表示的两个时间之间的差异，例如'hours'。't1'和't2'可以是Date或DateTime，如果指定'timezone'，它将应用于两个参数。如果不是，则使用来自数据类型't1'和't2'的时区。如果时区不相同，则结果将是未定义的。

支持的单位值：

单位

第二

分钟

小时

日

周

月

季

年

时隙 (开始时间，持续时间，[，大小])

它返回一个时间数组，其中包括从从«StartTime»开始到«StartTime + Duration 秒»内的所有符合«size» (以秒为单位)步长的时间点。其中«size»是一个可选参数，默认为1800。

例如，`timeSlots(toDateTime('2012-01-01 12:20:00'), 600) = [toDateTime ('2012-01-01 12:00:00') , toDateTime ('2012-01-01 12:30:00')]`。

这对于搜索在相应会话中综合浏览量是非常有用的。

formatDateTime (时间，格式[，时区])

函数根据给定的格式字符串来格式化时间。请注意：格式字符串必须是常量表达式，例如：单个结果列不能有多种格式字符串。

支持的格式修饰符：

(«Example» 列是对 2018-01-02 22:33:44 的格式化结果)

修饰符	产品描述	示例
%C	年除以100并截断为整数(00-99)	20
%d	月中的一天，零填充 (01-31)	02
%D	短MM/DD/YY日期，相当于%m/%d/%y	01/02/2018
%e	月中的一天，空格填充 (1-31)	2
%F	短YYYY-MM-DD日期，相当于%Y-%m-%d	2018-01-02
%H	24小时格式 (00-23)	22
%I	小时12h格式 (01-12)	10
%j	一年(001-366)	002
%m	月份为十进制数 (01-12)	01

%M 修饰符	产品描述(00-59)	示例
%n	换行符("\n")	
%p	AM或PM指定	PM
%R	24小时HH:MM时间，相当于%H:%M	22:33
%S	第二(00-59)	44
%t	水平制表符('t')	
%T	ISO8601时间格式(HH:MM:SS)，相当于%H:%M:%S	22:33:44
%u	ISO8601平日as编号，星期一为1(1-7)	2
%V	ISO8601周编号(01-53)	01
%w	周日为十进制数，周日为0(0-6)	2
%y	年份，最后两位数字 (00-99)	18
%Y	年	2018
%%	%符号	%

机器学习函数

evalMLMethod (预测)

使用拟合回归模型的预测请使用 `evalMLMethod` 函数。请参阅 `linearRegression` 中的链接。

随机线性回归

`stochasticLinearRegression` 聚合函数使用线性模型和MSE损失函数实现随机梯度下降法。使用 `evalMLMethod` 来预测新数据。

请参阅示例和注释 [此处](#)。

随机逻辑回归

`stochasticLogisticRegression` 聚合函数实现了二元分类问题的随机梯度下降法。使用 `evalMLMethod` 来预测新数据。请参阅示例和注释 [此处](#)。

条件函数

如果 (cond, 那么, 否则) , cond? 运算符然后 : else

如果 `cond != 0` 则返回 `then`，如果 `cond = 0` 则返回 `else`。

`cond` 必须是 `UInt8` 类型，`then` 和 `else` 必须存在最低的共同类型。

`then` 和 `else` 可以是 `NULL`

多

允许您在查询中更紧凑地编写 `CASE` 运算符。

```
multilf(cond_1 then_1 cond_2 then_2 else)
```

```
multilf(cond_1, then_1, cond_2, then_2...else)
```

参数：

- `cond_N` — 函数返回 `then_N` 的条件。
- `then_N` — 执行时函数的结果。
- `else` — 如果没有满足任何条件，则为函数的结果。

该函数接受 $2N + 1$ 参数。

返回值

该函数返回值《`then_N`》或《`else`》之一，具体取决于条件 `cond_N`。

示例

存在如下一张表

x	y
1	NULL
2	3

执行查询 `SELECT multilf(isNull(y), x, y < 3, y, NULL) FROM t_null`。结果：

multilf(isNull(y), x, less(y, 3), y, NULL)
1 NULL

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。

例如，您无法将日期与字符串进行比较。您必须使用函数将字符串转换为日期，反之亦然。

字符串按字节进行比较。较短的字符串小于以其开头并且至少包含一个字符的所有字符串。

注意。直到 1.1.54134 版本，有符号和无符号数字的比较方式与 C++ 相同。换句话说，在 `SELECT 9223372036854775807 > -1` 等情况下，您可能会得到错误的结果。此行为在版本 1.1.54134 中已更改，现在在数学上是正确的。

等于，`a=b` 和 `a==b` 运算符

`notEquals,a!` 运算符 `=b` 和 `a <> b`

少，`< operator`

更大，`> operator`

中相等级 `<- operator`

山丘之父，`~ operator`

伟大的等级，`>= operator`

算术函数

对于所有算术函数，结果类型为结果适合的最小数字类型（如果存在这样的类型）。最小数字类型是根据数字的位数，是否有符号以及是否是浮点类型而同时进行的。如果没有足够的位，则采用最高位类型。

例如：

```
SELECT toTypeName(0), toTypeName(0 + 0), toTypeName(0 + 0 + 0), toTypeName(0 + 0 + 0 + 0)
```

toTypeName(0)	toTypeName(plus(0, 0))	toTypeName(plus(plus(0, 0), 0))	toTypeName(plus(plus(plus(0, 0), 0), 0))
UInt8	UInt16	UInt32	UInt64

算术函数适用于UInt8，UInt16，UInt32，UInt64，Int8，Int16，Int32，Int64，Float32或Float64中的任何类型。

溢出的产生方式与C++相同。

加(a,b),a+b

计算数字的总和。

您还可以将Date或DateTime与整数进行相加。在Date的情况下，添加的整数意味着添加相应的天数。对于DateTime，这意味着这添加相应的描述。

减(a,b),a-b

计算数字之间的差，结果总是有符号的。

您还可以将Date或DateTime与整数进行相减。见上面的'plus'。

乘(a,b),a*b

计算数字的乘积。

除以(a,b),a/b

计算数字的商。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'。

intDiv(a,b)

计算整数数字的商，向下舍入（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。

模(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。

否定(a),-a

计算一个数字的

用反转符号计算一个数字。结果始终是签名的。

计算具有反向符号的数字。结果始终签名。

abs(a)

计算数字 (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)

返回数字的最大公约数。

除以零或将最小负数除以-1时抛出异常。

lcm(a,b)

返回数字的最小公倍数。

除以零或将最小负数除以-1时抛出异常。

类型转换函数

toUInt8,toUInt16,toUInt32,toUInt64

toInt8,toInt16,toInt32,toInt64

toFloat32,toFloat64

今天，今天

toUInt8OrZero,toUInt16OrZero,toUInt32OrZero,toUInt64OrZero

toUInt8OrNull,toUInt16OrNull,toUInt32OrNull,toUInt64OrNull,to

toString

这些函数用于在数字、字符串（不包含FixedString）、Date以及DateTime之间互相转换。

所有的函数都接受一个参数。

当将其他类型转换到字符串或从字符串转换到其他类型时，使用与TabSeparated格式相同的规则对字符串的值进行格式化或解析。如果无法解析字符串则抛出异常并取消查询。

当将Date转换为数字或反之，Date对应Unix时间戳的天数。

将DateTime转换为数字或反之，DateTime对应Unix时间戳的秒数。

toDate/toDateTime函数的日期和日期时间格式定义如下：

```
YYYY-MM-DD  
YYYY-MM-DD hh:mm:ss
```

例外的是，如果将UInt32、Int32、UInt64或Int64类型的数值转换为Date类型，并且其对应的值大于等于65536，则该数值将被解析成unix时间戳（而不是对应的天数）。这意味着允许写入'toDate(unix_timestamp)'这种常见情况，否则这将是错误的，并且需要便携更加繁琐的'toDate(toDateTime(unix_timestamp))'。

Date与DateTime之间的转换以更为自然的方式进行：通过添加空的time或删除time。

数值类型之间的转换与C++中不同数字类型之间的赋值相同的规则。

此外，`DateTime`参数的`toString`函数可以在第二个参数中包含时区名称。例如：`Asia/Yekaterinburg`在这种情况下，时间根据指定的时区进行格式化。

```
SELECT
    now() AS now_local,
    toString(now(), 'Asia/Yekaterinburg') AS now_yekat
```

now_local	now_yekat
2016-06-15 00:11:21	2016-06-15 02:11:21

另请参阅 `toUnixTimestamp` 函数。

`toDecimal32(value,S),toDecimal64(value,S),toDecimal128(valu`

将 `value` 转换为精度为 `S` 的十进制。`value` 可以是数字或字符串。`S` 参数为指定的小数位数。

`toFixedString(s,N)`

将 `String` 类型的参数转换为 `FixedString(N)` 类型的值（具有固定长度 `N` 的字符串）。`N` 必须是一个常量。

如果字符串的字节数少于 `N`，则向右填充空字节。如果字符串的字节数多于 `N`，则抛出异常。

`toStringCutToZero(s)`

接受 `String` 或 `FixedString` 参数。返回 `String`，其内容在找到的第一个零字节处被截断。

示例：

```
SELECT toFixedString('foo', 8) AS s, toStringCutToZero(s) AS s_cut
```

s	s_cut
foo\0\0\0\0\0	foo

```
SELECT toFixedString('foo\0bar', 8) AS s, toStringCutToZero(s) AS s_cut
```

s	s_cut
foo\0bar\0	foo

`reinterpretAsUInt8,reinterpretAsUInt16,reinterpretAsUInt32,reinterpretAsInt8,reinterpretAsInt16,reinterpretAsInt32,reinterpretAsFloat32,reinterpretAsFloat64`

重新解释日期，重新解释日期时间

这些函数接受一个字符串，并将放在字符串开头的字节解释为主机顺序中的数字（little endian）。如果字符串不够长，则函数就像使用必要数量的空字节填充字符串一样。如果字符串比需要的长，则忽略额外的字节。`Date` 被解释为 Unix 时间戳的天数，`DateTime` 被解释为 Unix 时间戳。

重新解释字符串

此函数接受数字、`Date` 或 `DateTime`，并返回一个字符串，其中包含表示主机顺序（小端）的相应值的字节。从末尾删除空

字节。例如，UInt32类型值255是一个字节长的字符串。

reinterpretAsFixedString

此函数接受数字、Date或DateTime，并返回包含表示主机顺序（小端）的相应值的字节的FixedString。从末尾删除空字节。例如，UInt32类型值255是一个长度为一个字节的FixedString。

演员(x,t)

将'x'转换为't'数据类型。还支持语法CAST (x AS t)

示例：

```
SELECT  
    '2016-06-15 23:00:00' AS timestamp,  
    CAST(timestamp AS DateTime) AS datetime,  
    CAST(timestamp AS Date) AS date,  
    CAST(timestamp, 'String') AS string,  
    CAST(timestamp, 'FixedString(22)') AS fixed_string
```

timestamp	datetime	date	string	fixed_string
2016-06-15 23:00:00	2016-06-15 23:00:00	2016-06-15	2016-06-15 23:00:00	2016-06-15 23:00:00\0\0\0

将参数转换为FixedString(N)，仅适用于String或FixedString(N)类型的参数。

支持将数据转换为可为空。例如：

```
SELECT toTypeName(x) FROM t_null
```

```
└─toTypeName(x)─┘  
| Int8 |  
| Int8 |  
|
```

```
SELECT toTypeName(CAST(x, 'Nullable(UInt16)')) FROM t_null
```

```
└─toTypeName(CAST(x, 'Nullable(UInt16)')) ─  
| Nullable(UInt16) ┌─  
| Nullable(UInt16) ┌─
```

将数字类型参数转换为Interval类型（时间区间）。

Interval类型实际上是非常有用的，您可以使用此类型的数据直接与Date或DateTime执行算术运算。同时，ClickHouse为Interval类型数据的声明提供了更方便的语法。例如：

```
WITH
    toDate('2019-01-01') AS date,
    INTERVAL 1 WEEK AS interval_week,
```

```
toIntervalWeek(1) AS interval_to_week  
SELECT  
    date + interval_week,  
    date + interval_to_week
```

```
+-----+-----+  
| plus(date, interval_week) | plus(date, interval_to_week) |  
| 2019-01-08 | 2019-01-08 |
```

parsedatetimebestefort

将数字类型参数解析为 Date 或 DateTime 类型。

与 toDate 和 toDateTime 不同，parseDateTimeBestEffort 可以进行更复杂的日期格式。

有关详细信息，请参阅链接：[复杂日期格式](#)。

parsedatetimebestefortornull

与 parsedatetimebestefort 相同，但它遇到无法处理的日期格式时返回 null。

parsedatetimebestefortorzero

与 parsedatetimebestefort 相同，但它遇到无法处理的日期格式时返回零 Date 或零 DateTime。

编码函数

hex

接受 String, unsigned integer, Date 或 DateTime 类型的参数。返回包含参数的十六进制表示的字符串。使用大写字母 A-F。不使用 0x 前缀或 h 后缀。对于字符串，所有字节都简单地编码为两个十六进制数字。数字转换为大端（《易阅读》）格式。对于数字，去除其中较旧的零，但仅限整个字节。例如，hex (1) ='01'。Date 被编码为自 Unix 时间开始以来的天数。DateTime 编码为自 Unix 时间开始以来的秒数。

unhex(str)

接受包含任意数量的十六进制数字的字符串，并返回包含相应字节的字符串。支持大写和小写字母 A-F。十六进制数字的数量不必是偶数。如果是奇数，则最后一位数被解释为 00-0F 字节的低位。如果参数字符串包含除十六进制数字以外的任何内容，则返回一些实现定义的结果（不抛出异常）。

如果要将结果转换为数字，可以使用《reverse》和《reinterpretAsType》函数。

UUIDStringToNum(str)

接受包含 36 个字符的字符串，格式为 «123e4567-e89b-12d3-a456-426655440000»，并将其转化为 FixedString (16) 返回。

UUIDNumToString(str)

接受 FixedString (16) 值。返回包含 36 个字符的文本格式的字符串。

位掩码列表(num)

接受一个整数。返回一个字符串，其中包含一组 2 的幂列表，其列表中的所有值相加等于这个整数。列表使用逗号分割，按升序排列。

位掩码阵列(num)

接受一个整数。返回一个 UInt64 类型数组，其中包含一组 2 的幂列表，其列表中的所有值相加等于这个整数。数组中的数字按升序排列。

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，和运营商
或，或运营商
不是，不是运营商
异或

随机函数

随机函数使用非加密方式生成伪随机数字。

所有随机函数都只接受一个参数或不接受任何参数。

您可以向它传递任何类型的参数，但传递的参数将不会使用在任何随机数生成过程中。

此参数的唯一目的是防止公共子表达式消除，以便在相同的查询中使用相同的随机函数生成不同的随机数。

兰德

返回一个UInt32类型的随机数字，所有UInt32类型的数字被生成的概率均相等。此函数线性同于的方式生成随机数。

rand64

返回一个UInt64类型的随机数字，所有UInt64类型的数字被生成的概率均相等。此函数线性同于的方式生成随机数。

randConstant

返回一个UInt32类型的随机数字，该函数不同之处在于仅为每个数据块参数一个随机数。

高阶函数

-> 运算符, lambda(params, expr) 函数

用于描述一个lambda函数用来传递给其他高阶函数。箭头的左侧有一个形式参数，它可以是一个标识符或多个标识符所组成的元祖。箭头的右侧是一个表达式，在这个表达式中可以使用形式参数列表中的任何一个标识符或表的任何一个列名。

示例: `x -> 2 * x, str -> str != Referer.`

高阶函数只能接受lambda函数作为其参数。

高阶函数可以接受多个参数的lambda函数作为其参数，在这种情况下，高阶函数需要同时传递几个长度相等的数组，这些数组将被传递给lambda参数。

除了'arrayMap'和'arrayFilter'以外的所有其他函数，都可以省略第一个参数（lambda函数）。在这种情况下，默认返回数组元素本身。

arrayMap(func, arr1, ...)

将arr

将从'func'函数的原始应用程序获得的数组返回到'arr'数组中的每个元素。

返回从原始应用程序获得的数组 'func' 函数中的每个元素 'arr' 阵列。

arrayFilter(func, arr1, ...)

返回一个仅包含以下元素的数组 'arr1' 对于哪个 'func' 返回0以外的内容。

示例:

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

```
└─res
  └─['abc World']
```

```
SELECT
arrayFilter(
  (i, x) -> x LIKE '%World%',
  arrayEnumerate(arr),
  ['Hello', 'abc World'] AS arr)
AS res
```

```
└─res
  └─[2]
```

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。

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
```

```
└─res
  └─[1, 2, 3, 4]
```

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。例如：

```
SELECT arrayCumSumNonNegative([1, 1, -4, 1]) AS res
```

```
└── res
  └── [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。

Schwartzian变换用于提高排序效率。

示例：

```
SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1]);
```

```
└── res
  └── ['world', 'hello']
```

请注意，`NULL`和`NaN`在最后（`NaN`在`NULL`之前）。例如：

```
SELECT arraySort([1, nan, 2, NULL, 3, nan, 4, NULL])
```

```
└── arraySort([1, nan, 2, NULL, 3, nan, 4, NULL])
  └── [1,2,3,4,NaN,NaN,NULL,NULL]
```

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。

请注意，`NULL`和`NaN`在最后（`NaN`在`NULL`之前）。例如：

```
SELECT arrayReverseSort([1, nan, 2, NULL, 3, nan, 4, NULL])
```

```
└── arrayReverseSort([1, nan, 2, NULL, 3, nan, 4, NULL])
  └── [4,3,2,1,NaN,NaN,NULL,NULL]
```

IN Operators

The `IN`, `NOT IN`, `GLOBAL IN`, and `GLOBAL NOT IN` operators are covered separately, since their functionality is quite rich.

The left side of the operator is either a single column or a tuple.

Examples:

```
SELECT UserID IN (123, 456) FROM ...
SELECT (CounterID, UserID) IN ((34, 123), (101500, 456)) FROM ...
```

If the left side is a single column that is in the index, and the right side is a set of constants, the system uses the index for processing the query.

Don't list too many values explicitly (i.e. millions). If a data set is large, put it in a temporary table (for example, see the section "External data for query processing"), then use a subquery.

The right side of the operator can be a set of constant expressions, a set of tuples with constant expressions (shown in the examples above), or the name of a database table or SELECT subquery in brackets.

If the right side of the operator is the name of a table (for example, `UserID IN users`), this is equivalent to the subquery `UserID IN (SELECT * FROM users)`. Use this when working with external data that is sent along with the query. For example, the query can be sent together with a set of user IDs loaded to the 'users' temporary table, which should be filtered.

If the right side of the operator is a table name that has the Set engine (a prepared data set that is always in RAM), the data set will not be created over again for each query.

The subquery may specify more than one column for filtering tuples.

Example:

```
SELECT (CounterID, UserID) IN (SELECT CounterID, UserID FROM ...)
```

The columns to the left and right of the IN operator should have the same type.

The IN operator and subquery may occur in any part of the query, including in aggregate functions and lambda functions.

Example:

```
SELECT
    EventDate,
    avg(UserID) IN
    (
        SELECT UserID
        FROM test.hits
        WHERE EventDate = toDate('2014-03-17')
    ) AS ratio
FROM test.hits
GROUP BY EventDate
ORDER BY EventDate ASC
```

EventDate	ratio
2014-03-17	1
2014-03-18	0.807696
2014-03-19	0.755406
2014-03-20	0.723218
2014-03-21	0.697021
2014-03-22	0.647851
2014-03-23	0.648416

For each day after March 17th, count the percentage of pageviews made by users who visited the site on March 17th.

A subquery in the IN clause is always run just one time on a single server. There are no dependent subqueries.

NULL Processing

During request processing, the IN operator assumes that the result of an operation with `NULL` is always equal

During request processing, the IN operator assumes that the result of an operation with **NULL** is always equal to 0, regardless of whether **NULL** is on the right or left side of the operator. **NULL** values are not included in any dataset, do not correspond to each other and cannot be compared.

Here is an example with the `t_null` table:

x	y
1	NULL
2	3

Running the query `SELECT x FROM t_null WHERE y IN (NULL,3)` gives you the following result:

x
2

You can see that the row in which $y = \text{NULL}$ is thrown out of the query results. This is because ClickHouse can't decide whether **NULL** is included in the `(NULL,3)` set, returns 0 as the result of the operation, and `SELECT` excludes this row from the final output.

```
SELECT y IN (NULL, 3)
FROM t_null
```

in(y, tuple(NULL, 3))
0
1

Distributed Subqueries

There are two options for IN-s with subqueries (similar to JOINs): normal IN / JOIN and GLOBAL IN / GLOBAL JOIN. They differ in how they are run for distributed query processing.

Attention

Remember that the algorithms described below may work differently depending on the `settings distributed_product_mode` setting.

When using the regular IN, the query is sent to remote servers, and each of them runs the subqueries in the IN or JOIN clause.

When using GLOBAL IN / GLOBAL JOINs, first all the subqueries are run for GLOBAL IN / GLOBAL JOINs, and the results are collected in temporary tables. Then the temporary tables are sent to each remote server, where the queries are run using this temporary data.

For a non-distributed query, use the regular IN / JOIN.

Be careful when using subqueries in the IN / JOIN clauses for distributed query processing.

Let's look at some examples. Assume that each server in the cluster has a normal **local_table**. Each server also has a **distributed_table** table with the **Distributed** type, which looks at all the servers in the cluster.

For a query to the **distributed_table**, the query will be sent to all the remote servers and run on them using the **local_table**.

For example, the query

```
SELECT uniq(UserID) FROM distributed_table
```

will be sent to all remote servers as

```
SELECT uniq(UserID) FROM local_table
```

and run on each of them in parallel, until it reaches the stage where intermediate results can be combined. Then the intermediate results will be returned to the requestor server and merged on it, and the final result will be sent to the client.

Now let's examine a query with IN:

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM local_table WHERE CounterID = 34)
```

- Calculation of the intersection of audiences of two sites.

This query will be sent to all remote servers as

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM local_table WHERE CounterID = 34)
```

In other words, the data set in the IN clause will be collected on each server independently, only across the data that is stored locally on each of the servers.

This will work correctly and optimally if you are prepared for this case and have spread data across the cluster servers such that the data for a single UserID resides entirely on a single server. In this case, all the necessary data will be available locally on each server. Otherwise, the result will be inaccurate. We refer to this variation of the query as "local IN".

To correct how the query works when data is spread randomly across the cluster servers, you could specify **distributed_table** inside a subquery. The query would look like this:

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

This query will be sent to all remote servers as

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

The subquery will begin running on each remote server. Since the subquery uses a distributed table, the subquery that is on each remote server will be resent to every remote server as

```
SELECT UserID FROM local_table WHERE CounterID = 34
```

For example, if you have a cluster of 100 servers, executing the entire query will require 10,000 elementary requests, which is generally considered unacceptable.

In such cases, you should always use GLOBAL IN instead of IN. Let's look at how it works for the query

In such cases, you should always use GLOBAL IN instead of IN. Let's look at how it works for the query

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID GLOBAL IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

The requestor server will run the subquery

```
SELECT UserID FROM distributed_table WHERE CounterID = 34
```

and the result will be put in a temporary table in RAM. Then the request will be sent to each remote server as

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID GLOBAL IN _data1
```

and the temporary table `_data1` will be sent to every remote server with the query (the name of the temporary table is implementation-defined).

This is more optimal than using the normal IN. However, keep the following points in mind:

1. When creating a temporary table, data is not made unique. To reduce the volume of data transmitted over the network, specify DISTINCT in the subquery. (You don't need to do this for a normal IN.)
2. The temporary table will be sent to all the remote servers. Transmission does not account for network topology. For example, if 10 remote servers reside in a datacenter that is very remote in relation to the requestor server, the data will be sent 10 times over the channel to the remote datacenter. Try to avoid large data sets when using GLOBAL IN.
3. When transmitting data to remote servers, restrictions on network bandwidth are not configurable. You might overload the network.
4. Try to distribute data across servers so that you don't need to use GLOBAL IN on a regular basis.
5. If you need to use GLOBAL IN often, plan the location of the ClickHouse cluster so that a single group of replicas resides in no more than one data center with a fast network between them, so that a query can be processed entirely within a single data center.

It also makes sense to specify a local table in the `GLOBAL IN` clause, in case this local table is only available on the requestor server and you want to use data from it on remote servers.

操作符

所有的操作符（运算符）都会在查询时依据他们的优先级及其结合顺序在被解析时转换为对应的函数。下面按优先级从高到低列出各组运算符及其对应的函数：

下标运算符

`a[N]` - 数组中的第N个元素；对应函数 `arrayElement(a, N)`

`a.N` - 元组中第N个元素；对应函数 `tupleElement(a, N)`

负号

`-a` - 对应函数 `negate(a)`

乘号、除号和取余

`a * b` - 对应函数 `multiply(a, b)`

`a / b` - 对应函数 `divide(a, b)`

`a % b` - 对应函数 `modulo(a, b)`

加号和减号

`a + b` - 对应函数 `plus(a, b)`

`a - b` - 对应函数 `minus(a, b)`

关系运算符

`a = b` - 对应函数 `equals(a, b)`

`a == b` - 对应函数 `equals(a, b)`

`a != b` - 对应函数 `notEquals(a, b)`

`a <> b` - 对应函数 `notEquals(a, b)`

`a <= b` - 对应函数 `lessOrEquals(a, b)`

`a >= b` - 对应函数 `greaterOrEquals(a, b)`

`a < b` - 对应函数 `less(a, b)`

`a > b` - 对应函数 `greater(a, b)`

`a LIKE s` - 对应函数 `like(a, b)`

`a NOT LIKE s` - 对应函数 `notLike(a, b)`

`a BETWEEN b AND c` - 等价于 `a >= b AND a <= c`

集合关系运算符

详见此节 **IN** 相关操作符。

`a IN ...` - 对应函数 `in(a, b)`

`a NOT IN ...` - 对应函数 `notIn(a, b)`

`a GLOBAL IN ...` - 对应函数 `globalIn(a, b)`

`a GLOBAL NOT IN ...` - 对应函数 `globalNotIn(a, b)`

逻辑非

`NOT a` - 对应函数 `not(a)`

逻辑与

`a AND b` - 对应函数 `and(a, b)`

逻辑或

`a OR b` - 对应函数 `or(a, b)`

条件运算符

`a ? b : c` - 对应函数 `if(a, b, c)`

注意：

条件运算符会先计算表达式`b`和表达式`c`的值，再根据表达式`a`的真假，返回相应的值。如果表达式`b`和表达式`c`是 **arrayJoin()** 函数，则不管表达式`a`是真是假，每行都会被复制展开。

使用日期和时间的操作员

EXTRACT

```
EXTRACT(part FROM date);
```

从给定日期中提取部件。例如，您可以从给定日期检索一个月，或从时间检索一秒钟。

该 `part` 参数指定要检索的日期部分。以下值可用：

- `DAY` — The day of the month. Possible values: 1-31.
- `MONTH` — The number of a month. Possible values: 1-12.
- `YEAR` — The year.
- `SECOND` — The second. Possible values: 0-59.
- `MINUTE` — The minute. Possible values: 0-59.
- `HOUR` — The hour. Possible values: 0-23.

该 `part` 参数不区分大小写。

该 `date` 参数指定要处理的日期或时间。无论是 [日期](#) 或 [日期时间](#) 支持类型。

例：

```
SELECT EXTRACT(DAY FROM toDate('2017-06-15'));
SELECT EXTRACT(MONTH FROM toDate('2017-06-15'));
SELECT EXTRACT(YEAR FROM toDate('2017-06-15'));
```

在下面的例子中，我们创建一个表，并在其中插入一个值 `DateTime` 类型。

```
CREATE TABLE test.Orders
(
    OrderId UInt64,
    OrderName String,
    OrderDate DateTime
)
ENGINE = Log;
```

```
INSERT INTO test.Orders VALUES (1, 'Jarlsberg Cheese', toDateTime('2008-10-11 13:23:44'));
```

```
SELECT
    toYear(OrderDate) AS OrderYear,
    toMonth(OrderDate) AS OrderMonth,
    toDayOfMonth(OrderDate) AS OrderDay,
    toHour(OrderDate) AS OrderHour,
    toMinute(OrderDate) AS OrderMinute,
    toSecond(OrderDate) AS OrderSecond
FROM test.Orders;
```

OrderYear	OrderMonth	OrderDay	OrderHour	OrderMinute	OrderSecond
2008	10	11	13	23	44

你可以看到更多的例子 [测试](#).

INTERVAL

创建一个 `间隔`-对应的不还具中使用的类型值 `日期` 和 `日期时间`-类型值。

示例：

```
SELECT now() AS current_date_time, current_date_time + INTERVAL 4 DAY + INTERVAL 3 HOUR
```

current_date_time	plus(plus(now(), toIntervalDay(4)), toIntervalHour(3))
2019-10-23 11:16:28	2019-10-27 14:16:28

另请参阅

- `间隔` 数据类型
- `toInterval` 类型转换函数

CASE 条件表达式

```
CASE [x]
WHEN a THEN b
[WHEN ... THEN ...]
[ELSE c]
END
```

如果指定了 `x`，该表达式会转换为 `transform(x, [a, ...], [b, ...], c)` 函数。否则转换为 `multilf(a, b, ..., c)`

如果该表达式中没有 `ELSE c` 子句，则默认值就是 `NULL`

但 `transform` 函数不支持 `NULL`

连接运算符

`s1 || s2` - 对应函数 `concat(s1, s2)`

创建 Lambda 函数

`x -> expr` - 对应函数 `lambda(x, expr)`

接下来的这些操作符因为其本身是括号没有优先级：

创建数组

`[x1, ...]` - 对应函数 `array(x1, ...)`

创建元组

`(x1, x2, ...)` - 对应函数 `tuple(x2, x2, ...)`

结合方式

所有的同级操作符从左到右结合。例如，`1 + 2 + 3` 会转换成 `plus(plus(1, 2), 3)`。

所以，有时他们会跟我们预期的不太一样。例如，`SELECT 4 > 2 > 3` 的结果是0。

为了高效，`and` 和 `or` 函数支持任意多参数，一连串的 `AND` 和 `OR` 运算符会转换成其对应的单个函数。

判断是否为 `NULL`

ClickHouse 支持 `IS NULL` 和 `IS NOT NULL`。

IS NULL

- 对于 可为空 类型的值，`IS NULL` 会返回：
 - 1 值为 `NULL`
 - 0 否则
- 对于其他类型的值，`IS NULL` 总会返回 0

```
:) SELECT x+100 FROM t_null WHERE y IS NULL
```

```
SELECT x + 100  
FROM t_null  
WHEREisNull(y)
```

```
└─plus(x, 100)─  
   ┌─┐  
   101 |  
   └─┘
```

1 rows in set. Elapsed: 0.002 sec.

IS NOT NULL

- 对于 可为空 类型的值，`IS NOT NULL` 会返回：
 - 0 值为 `NULL`
 - 1 否则
- 对于其他类型的值，`IS NOT NULL` 总会返回 1

```
:) SELECT * FROM t_null WHERE y IS NOT NULL
```

```
SELECT *  
FROM t_null  
WHERE isNotNull(y)
```

```
└─x ─y─  
   ┌─┐  
   2 | 3 |  
   └─┘
```

1 rows in set. Elapsed: 0.002 sec.

UUID

通用唯一标识符(UUID)是用于标识记录的16字节数。有关UUID的详细信息，请参阅 [维基百科](#).

UUID类型值的示例如下所示：

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

如果在插入新记录时未指定UUID列值，则UUID值将用零填充：

```
00000000-0000-0000-0000-000000000000
```

如何生成

要生成UUID值，ClickHouse提供了 [generateuidv4](#) 功能。

用法示例

示例 1

此示例演示如何创建具有UUID类型列的表并将值插入到表中。

```
CREATE TABLE t_uuid (x UUID, y String) ENGINE=TinyLog
```

```
INSERT INTO t_uuid SELECT generateUUIDv4(), 'Example 1'
```

```
SELECT * FROM t_uuid
```

x	y
417ddc5d-e556-4d27-95dd-a34d84e46a50	Example 1

示例2

在此示例中，插入新记录时未指定UUID列值。

```
INSERT INTO t_uuid (y) VALUES ('Example 2')
```

```
SELECT * FROM t_uuid
```

x	y
417ddc5d-e556-4d27-95dd-a34d84e46a50	Example 1
00000000-0000-0000-0000-000000000000	Example 2

限制

UUID数据类型仅支持以下功能 **字符串** 数据类型也支持（例如, **min**, **max**，和 **计数**）.

算术运算不支持UUID数据类型（例如, **abs**）或聚合函数，例如 **sum** 和 **avg**.

Datetime64

允许存储时间instant间，可以表示为日历日期和一天中的时间，具有定义的亚秒精度

刻度尺寸（精度）： $10^{-\text{精度}}$ 秒

语法：

```
DateTime64(precision, [timezone])
```

在内部，存储数据作为一些‘ticks’自纪元开始(1970-01-01 00:00:00UTC)作为Int64. 刻度分辨率由precision参数确定。此外，该 **DateTime64** 类型可以存储时区是相同的整个列，影响如何的值 **DateTime64** 类型值以文本格式显示，以及如何解析指定为字符串的值('2020-01-01 05:00:01.000')。时区不存储在表的行中（或resultset中），而是存储在列元数据中。查看详细信息 [日期时间](#)。

例

1. 创建一个表 DateTime64-输入列并将数据插入其中:

```
CREATE TABLE dt
(
    `timestamp` DateTime64(3, 'Europe/Moscow'),
    `event_id` UInt8
)
ENGINE = TinyLog
```

```
INSERT INTO dt Values (1546300800000, 1), ('2019-01-01 00:00:00', 2)
```

```
SELECT * FROM dt
```

timestamp	event_id
2019-01-01 03:00:00.000	1
2019-01-01 00:00:00.000	2

- 将日期时间作为整数插入时，将其视为适当缩放的Unix时间戳(UTC)。`1546300800000`（精度为3）表示`'2019-01-01 00:00:00'` UTC. 然而，作为`timestamp`列有`Europe/Moscow`（UTC+3）指定的时区，当输出为字符串时，该值将显示为`'2019-01-01 03:00:00'`
- 当插入字符串值作为日期时间时，它被视为处于列时区。`'2019-01-01 00:00:00'` 将被视为`Europe/Moscow`时区并存储为`1546290000000`.

2. 过滤 DateTime64 值

```
SELECT * FROM dt WHERE timestamp = toDateDateTime64('2019-01-01 00:00:00', 3, 'Europe/Moscow')
```

timestamp	event_id
2019-01-01 00:00:00.000	2

不像`DateTime`, `DateTime64` 值不转换为`String`自动

3. 获取一个时区 DateTime64-类型值:

```
SELECT toDateDateTime64(now(), 3, 'Europe/Moscow') AS column, toTypeName(column) AS x
```

column	x
2019-10-16 04:12:04.000	<code>DateTime64(3, 'Europe/Moscow')</code>

4. 时区转换

```
SELECT
toDateDateTime64(timestamp, 3, 'Europe/London') as lon_time,
toDateDateTime64(timestamp, 3, 'Europe/Moscow') as mos_time
FROM dt
```

lon_time	mos_time
2019-01-01 00:00:00.000	2019-01-01 03:00:00.000
2018-12-31 21:00:00.000	2019-01-01 00:00:00.000

另请参阅

- 类型转换函数
- 用于处理日期和时间的函数
- 用于处理数组的函数
- 该 `date_time_input_format` 设置
- 该 `timezone` 服务器配置参数
- 使用日期和时间的操作员
- `Date` 数据类型
- `DateTime` 数据类型

IPv4

`IPv4` 是与 `UInt32` 类型保持二进制兼容的 `Domain` 类型，其用于存储 IPv4 地址的值。它提供了更为紧凑的二进制存储的同时支持识别可读性更加友好的输入输出格式。

基本使用

```
CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY url;
```

```
DESCRIBE TABLE hits;
```

name	type	default_type	default_expression	comment	codec_expression
url	String				
from	IPv4				

同时您也可以使用 `IPv4` 类型的列作为主键：

```
CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY from;
```

在写入与查询时，`IPv4` 类型能够识别可读性更加友好的输入输出格式：

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '116.253.40.133')('https://clickhouse.tech', '183.247.232.58')('https://clickhouse.yandex/docs/en/', '116.106.34.242');
```

```
SELECT * FROM hits;
```

url	from
https://clickhouse.tech/docs/en/	116.106.34.242
https://wikipedia.org	116.253.40.133
https://clickhouse.tech	183.247.232.58

同时它提供更为紧凑的二进制存储格式：

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

```
└─toTypeName(from)─┐hex(from)─┐  
| IPv4           | B7F7E83A |
```

不可隐式转换为除 UInt32 以外的其他类型。如果要将 IPv4 类型的值转换成字符串，你可以使用 `IPv4NumToString()` 显示的进行转换：

```
SELECT toTypeName(s), IPv4NumToString(from) as s FROM hits LIMIT 1;
```

```
└─toTypeName(IPv4NumToString(from))─┐s─┐  
| String            | 183.247.232.58 |
```

或可以使用 `CAST` 将它转换为 UInt32 类型：

```
SELECT toTypeName(i), CAST(from as UInt32) as i FROM hits LIMIT 1;
```

```
└─toTypeName(CAST(from, 'UInt32'))─┐i─┐  
| UInt32          | 3086477370 |
```

IPv6

IPv6 是与 `FixedString(16)` 类型保持二进制兼容的 `Domain` 类型，其用于存储 IPv6 地址的值。它提供了更为紧凑的二进制存储的同时支持识别可读性更加友好的输入输出格式。

基本用法

```
CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY url;
```

```
DESCRIBE TABLE hits;
```

```
└─name─type─default_type─default_expression─comment─codec_expression─  
| url | String |           |           |           |           |  
| from | IPv6  |           |           |           |           |
```

同时您也可以使用 IPv6 类型的列作为主键：

```
CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY from;
```

在写入与查询时，IPv6 类型能够识别可读性更加友好的输入输出格式：

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '2a02:aa08:e000:3100::2')('https://clickhouse.tech',  
'2001:44c8:129:2632:33:0:252:2')('https://clickhouse.yandex/docs/en/', '2a02:e980:1e::1');
```

```
SELECT * FROM hits;
```

```
└─url─────────from─────────┘
```

```
| https://clickhouse.tech | 2001:44c8:129:2632:33:0:252:2 |
| https://clickhouse.tech/docs/en/ | 2a02:e980:1e::1 |
| https://wikipedia.org | 2a02:aa08:e000:3100::2
```

同时它提供更为紧凑的二进制存储格式：

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

```
└─toTypeName(from)─┐hex(from)─┐
| IPv6           | 200144C801292632003300002520002 |
```

不可隐式转换为除 `FixedString(16)` 以外的其他类型。如果要将 `IPv6` 类型的值转换成字符串，你可以使用 `IPv6NumToString()` 显示的进行转换：

```
SELECT toTypeName(s), IPv6NumToString(from) as s FROM hits LIMIT 1;
```

```
└─toTypeName(IPv6NumToString(from))─┐s─┐
| String                         | 2001:44c8:129:2632:33:0:252:2 |
```

或使用 `CAST` 将其转换为 `FixedString(16)`：

```
SELECT toTypeName(i), CAST(from as FixedString(16)) as i FROM hits LIMIT 1;
```

```
└─toTypeName(CAST(from, 'FixedString(16)'))─┐i─┐
| FixedString(16)                         | ♦♦♦ |
```

域

`Domain` 类型是特定实现的类型，它总是与某个现存的基础类型保持二进制兼容的同时添加一些额外的特性，以能够在维持磁盘数据不变的情况下使用这些额外的特性。目前 ClickHouse 暂不支持自定义 `domain` 类型。

如果你可以在一个地方使用与 `Domain` 类型二进制兼容的基础类型，那么在相同的地方您也可以使用 `Domain` 类型，例如：

- 使用 `Domain` 类型作为表中列的类型
- 对 `Domain` 类型的列进行读/写数据
- 如果与 `Domain` 二进制兼容的基础类型可以作为索引，那么 `Domain` 类型也可以作为索引
- 将 `Domain` 类型作为参数传递给函数使用
- 其他

Domains 的额外特性

- 在执行 `SHOW CREATE TABLE` 或 `DESCRIBE TABLE` 时，其对应的列总是展示为 `Domain` 类型的名称
- 在 `INSERT INTO domain_table(domain_column) VALUES(...)` 中输入数据总是以更人性化的格式进行输入
- 在 `SELECT domain_column FROM domain_table` 中数据总是以更人性化的格式输出
- 在 `INSERT INTO domain_table FORMAT CSV ...` 中，实现外部源数据以更人性化的格式载入

Domains 类型的限制

- 无法通过 `ALTER TABLE` 将基础类型的索引转换为 `Domain` 类型的索引。

- 当从其他列或表插入数据时，无法将string类型的值隐式地转换为Domain类型的值。
- 无法对存储为Domain类型的值添加约束。

AggregateFunction(name, types_of_arguments...)

聚合函数的中间状态，可以通过聚合函数名称加-`State`后缀的形式得到它。与此同时，当您需要访问该类型的最终状态数据时，您需要以相同的聚合函数名加-`Merge`后缀的形式来得到最终状态数据。

`AggregateFunction` — 参数化的数据类型。

参数

- 聚合函数名

如果函数具备多个参数列表，请在此处指定其他参数列表中的值。

- 聚合函数参数的类型

示例

```
CREATE TABLE t
(
    column1 AggregateFunction(uniq, UInt64),
    column2 AggregateFunction(anyIf, String, UInt8),
    column3 AggregateFunction(quantiles(0.5, 0.9), UInt64)
) ENGINE = ...
```

上述中的`uniq`，`anyIf`（任何+如果）以及 分位数 都为ClickHouse中支持的聚合函数。

使用指南

数据写入

当需要写入数据时，您需要将数据包含在`INSERT SELECT`语句中，同时对于`AggregateFunction`类型的数据，您需要使用对应的以-`State`为后缀的函数进行处理。

函数使用示例

```
uniqState(UserID)
quantilesState(0.5, 0.9)(SendTiming)
```

不同于`uniq`和`quantiles`函数返回聚合结果的最终值，以-`State`后缀的函数总是返回`AggregateFunction`类型的数据的中间状态。

对于`SELECT`而言，`AggregateFunction`类型总是以特定的二进制形式展现在所有的输出格式中。例如，您可以使用`SELECT`语句将函数的状态数据转储为`TabSeparated`格式的同时使用`INSERT`语句将数据转储回去。

数据查询

当从`AggregatingMergeTree`表中查询数据时，对于`AggregateFunction`类型的字段，您需要使用以-`Merge`为后缀的相同聚合函数来聚合数据。对于非`AggregateFunction`类型的字段，请将它们包含在`GROUP BY`子句中。

以-`Merge`为后缀的聚合函数，可以将多个`AggregateFunction`类型的中间状态组合计算为最终的聚合结果。

例如，如下的两个查询返回的结果总是一致：

```
SELECT uniq(UserID) FROM table
```

```
SELECT uniqMerge(state) FROM (SELECT uniqState(userID) AS state FROM table GROUP BY RegionID)
```

```
SELECT uniqueMergeState, FROM (SELECT uniqstate(USERTID) AS state FROM table GROUP BY regid);
```

使用示例

请参阅 [AggregatingMergeTree](#) 的说明

Decimal(P,S), Decimal32(S), Decimal64(S), Decimal128(S)

有符号的定点数，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）。

参数

- P - 精度。有效范围：[1:38]，决定可以有多少个十进制数字（包括分数）。
- S - 规模。有效范围：[0 : P]，决定数字的小数部分中包含的小数位数。

对于不同的 P 参数值 Decimal 表示，以下例子都是同义的：

- P从[1:9]-对于Decimal32(S)
- P从[10:18]-对于Decimal64(小号)
- P从[19:38]-对于Decimal128 (S)

十进制值范围

- Decimal32(S) - (-1 * 10^(9-S), 1*10^(9-S))
- Decimal64(S) - (-1 * 10^(18-S), 1*10^(18-S))
- Decimal128(S) - (-1 * 10^(38-S), 1*10^(38-S))

例如，Decimal32(4) 可以表示 -99999.9999 至 99999.9999 的数值，步长为 0.0001。

内部表示方式

数据采用与自身位宽相同的有符号整数存储。这个数在内存中实际范围会高于上述范围，从 String 转换到十进制数的时候会做对应的检查。

由于现代CPU不支持128位数字，因此 Decimal128 上的操作由软件模拟。所以 Decimal128 的运算速度明显慢于 Decimal32/Decimal64。

运算和结果类型

对Decimal的二进制运算导致更宽的结果类型（无论参数的顺序如何）。

- Decimal64(S1) <op> Decimal32(S2) -> Decimal64(S)
- Decimal128(S1) <op> Decimal32(S2) -> Decimal128(S)
- Decimal128(S1) <op> Decimal64(S2) -> Decimal128(S)

精度变化的规则：

- 加法，减法： $S = \max(S1, S2)$ 。
- 乘法： $S = S1 + S2$ 。
- 除法： $S = S1$ 。

对于 Decimal 和整数之间的类似操作，结果是与参数大小相同的十进制。

未定义Decimal和Float32/Float64之间的函数。要执行此类操作，您可以使用：toDecimal32、toDecimal64、toDecimal128 或 toFloat32，toFloat64，需要显式地转换其中一个参数。注意，结果将失去精度，类型转换是昂贵的操作。

Decimal上的一些函数返回结果为Float64（例如，var或stddev）。对于其中一些，中间计算发生在Decimal中。对于此类函数，尽管结果类型相同，但Float64和Decimal中相同数据的结果可能不同。

溢出检查

在对 Decimal 类型执行操作时，数值可能会发生溢出。分数中的过多数字被丢弃（不是舍入的）。整数中的过多数字将导致

异常。

```
SELECT toDecimal32(2, 4) AS x, x / 3
```

x	divide(toDecimal32(2, 4), 3)
2.0000	0.6666

```
SELECT toDecimal32(4.2, 8) AS x, x * x
```

DB::Exception: Scale is out of bounds.

```
SELECT toDecimal32(4.2, 8) AS x, 6 * x
```

DB::Exception: Decimal math overflow.

检查溢出会导致计算变慢。如果已知溢出不可能，则可以通过设置 `decimal_check_overflow` 来禁用溢出检查，在这种情况下，溢出将导致结果不正确：

```
SET decimal_check_overflow = 0;  
SELECT toDecimal32(4.2, 8) AS x, 6 * x
```

x	multiply(6, toDecimal32(4.2, 8))
4.20000000	-17.74967296

溢出检查不仅发生在算术运算上，还发生在比较运算上：

```
SELECT toDecimal32(1, 8) < 100
```

DB::Exception: Can't compare.

Enum8,Enum16

包括 `Enum8` 和 `Enum16` 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，但所有含有 `Enum` 数据类型的操作都是按照包含整数的值来执行。这在性能方面比使用 `String` 数据类型更有效。

- `Enum8` 用 `'String' = Int8` 对描述。
- `Enum16` 用 `'String' = Int16` 对描述。

用法示例

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'`。如果您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)

当您从表中查询数据时，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

在查询中创建枚举值，您还需要使用 `CAST`。

```
SELECT toTypeName(CAST('a', 'Enum8('a' = 1, 'b' = 2)'))
```

toTypeName(CAST('a', 'Enum8('a' = 1, 'b' = 2)'))
Enum8('a' = 1, 'b' = 2)

规则及用法

`Enum8` 类型的每个值范围是 `-128 ... 127`，`Enum16` 类型的每个值范围是 `-32768 ... 32767`。所有的字符串或者数字都必须是不一样的。允许存在空字符串。如果某个 `Enum` 类型被指定了（在表定义的时候），数字可以是任意顺序。然而，顺序并不重要。

`Enum` 中的字符串和数值都不能是 `NULL`。

`Enum` 包含在 `可为空` 类型中。因此，如果您使用此查询创建一个表

```
CREATE TABLE t_enum_nullable
(
    x Nullable( Enum8('hello' = 1, 'world' = 2) )
)
ENGINE = TinyLog
```

不仅可以存储 'hello' 和 'world'，还可以存储 `NULL`。

```
INSERT INTO t_enum_nullable Values('hello'),('world'),(NULL)
```

在内存中，`Enum` 列的存储方式与相应数值的 `Int8` 或 `Int16` 相同。

当以文本方式读取的时候，ClickHouse 将值解析成字符串然后去枚举值的集合中搜索对应字符串。如果没有找到，会抛出异常。当读取文本格式的时候，会根据读取到的字符串去找对应的数值。如果没有找到，会抛出异常。

当以文本形式写入时，ClickHouse 将值解析成字符串写入。如果列数据包含垃圾数据（不是来自有效集合的数字），则抛出异常。`Enum` 类型以二进制读取和写入的方式与 `Int8` 和 `Int16` 类型一样的。

隐式默认值是数值最小的值。

在 `ORDER BY`，`GROUP BY`，`IN`，`DISTINCT` 等等中，`Enum` 的行为与相应的数字相同。例如，按数字排序。对于等式运算符和比较运算符，`Enum` 的工作机制与它们在底层数值上的工作机制相同。

枚举值不能与数字进行比较。枚举可以与常量字符串进行比较。如果与之比较的字符串不是有效 `Enum` 值，则将引发异常。可以使用 `IN` 运算符来判断一个 `Enum` 是否存在于某个 `Enum` 集合中，其中集合中的 `Enum` 需要用字符串表示。

大多数具有数字和字符串的运算并不适用于 `Enums`；例如，`Enum` 类型不能和一个数值相加。但是，`Enum` 有一个原生的 `toString` 函数，它返回它的字符串值。

`Enum` 值使用 `toT` 函数可以转换成数值类型，其中 `T` 是一个数值类型。若 `T` 恰好对应 `Enum` 的底层数值类型，这个转换是零消耗的。

`Enum` 类型可以被 `ALTER` 无成本地修改对应集合的值。可以通过 `ALTER` 操作来增加或删除 `Enum` 的成员（只要表没有用到该值，删除都是安全的）。作为安全保障，改变之前使用过的 `Enum` 成员将抛出异常。

通过 `ALTER` 操作，可以将 `Enum8` 转成 `Enum16`，反之亦然，就像 `Int8` 转 `Int16` 一样。

Float32,Float64

浮点数。

类型与以下 C 语言中类型是相同的：

- `Float32` - `float`
- `Float64` - `double`

我们建议您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，例如货币数量或页面加载时间用毫秒为单位表示

使用浮点数

- 对浮点数进行计算可能引起四舍五入的误差。

```
SELECT 1 - 0.9
```

```
minus(1, 0.9)
0.09999999999999998 |
```

- 计算的结果取决于计算方法（计算机系统的处理器类型和体系结构）
- 浮点计算结果可能是诸如无穷大（`INF`）和«非数字»（`NaN`）。对浮点数计算的时候应该考虑到这点。
- 当一行行阅读浮点数的时候，浮点数的结果可能不是机器最近显示的数值。

南和Inf

与标准SQL相比，ClickHouse 支持以下类别的浮点数：

- Inf – 正无穷

```
SELECT 0.5 / 0
```

```
└─divide(0.5, 0)─  
    inf |
```

- -Inf – 负无穷

```
SELECT -0.5 / 0
```

```
└─divide(-0.5, 0)─  
    -inf |
```

- NaN – 非数字

```
SELECT 0 / 0
```

```
└─divide(0, 0)─  
    nan |
```

可以在 [ORDER BY 子句](#) 查看更多关于 NaN 排序的规则。

SimpleAggregateFunction

`SimpleAggregateFunction(name, types_of_arguments...)` data type stores current value of the aggregate function, and does not store its full state as `AggregateFunction` does. This optimization can be applied to functions for which the following property holds: the result of applying a function `f` to a row set `S1 UNION ALL S2` can be obtained by applying `f` to parts of the row set separately, and then again applying `f` to the results: `f(S1 UNION ALL S2) = f(f(S1) UNION ALL f(S2))`. This property guarantees that partial aggregation results are enough to compute the combined one, so we don't have to store and process any extra data.

The following aggregate functions are supported:

- any
- anyLast
- min
- max
- sum
- groupBitAnd
- groupBitOr
- groupBitXor

Values of the `SimpleAggregateFunction(func, Type)` look and stored the same way as `Type`, so you do not need to apply functions with `-Merge/-State` suffixes. `SimpleAggregateFunction` has better performance than `AggregateFunction` with same aggregation function.

Parameters

- Name of the aggregate function.
- Types of the aggregate function arguments.

Example

```
CREATE TABLE t
(
    column1 SimpleAggregateFunction(sum, UInt64),
    column2 SimpleAggregateFunction(any, String)
) ENGINE = ...
```

Tuple(T1, T2, ...)

元组，其中每个元素都有单独的 [类型](#)。

不能在表中存储元组（除了内存表）。它们可以用于临时列分组。在查询中，IN 表达式和带特定参数的 lambda 函数可以来对临时列进行分组。更多信息，请参阅 [IN 操作符](#) 和 [高阶函数](#)。

元组可以是查询的结果。在这种情况下，对于JSON以外的文本格式，括号中的值是逗号分隔的。在JSON格式中，元组作为数组输出（在方括号中）。

创建元组

可以使用函数来创建元组：

```
tuple(T1, T2, ...)
```

创建元组的示例：

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

```
SELECT
    (1, 'a') AS x,
    toTypeName(x)

    x-----toTypeName(tuple(1, 'a'))-----
    | (1,'a') | Tuple(UInt8, String) |
```

```
1 rows in set. Elapsed: 0.021 sec.
```

元组中的数据类型

在动态创建元组时，ClickHouse 会自动为元组的每一个参数赋予最小可表达的类型。如果参数为 [NULL](#)，那这个元组对应元素是 [可为空](#)。

自动数据类型检测示例：

```
SELECT tuple(1, NULL) AS x, toTypeName(x)
```

```
SELECT
    (1, NULL) AS x,
    toTypeName(x)

    x-----toTypeName(tuple(1, NULL))-----
```

```
| (1,NULL) | Tuple(UInt8, Nullable(Nothing)) |
```

1 rows in set. Elapsed: 0.002 sec.

UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64

固定长度的整型，包括有符号整型或无符号整型。

整型范围

- Int8-[-128:127]
- Int16-[-32768:32767]
- Int32-[-2147483648:2147483647]
- Int64-[-9223372036854775808:9223372036854775807]

无符号整型范围

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

可为空（类型名称）

允许用特殊标记 (NULL) 表示«缺失值»，可以与 `TypeName` 的正常值存放一起。例如，`Nullable(Int8)` 类型的列可以存储 `Int8` 类型值，而没有值的行将存储 `NULL`。

对于 `TypeName`，不能使用复合数据类型 [阵列](#) 和 [元组](#)。复合数据类型可以包含 `Nullable` 类型值，例如 `Array(Nullable(Int8))`。

`Nullable` 类型字段不能包含在表索引中。

除非在 ClickHouse 服务器配置中另有说明，否则 `NULL` 是任何 `Nullable` 类型的默认值。

存储特性

要在表的列中存储 `Nullable` 类型值，ClickHouse 除了使用带有值的普通文件外，还使用带有 `NUL` 掩码的单独文件。掩码文件中的条目允许 ClickHouse 区分每个表行的 `NULL` 和相应数据类型的默认值。由于附加了新文件，`Nullable` 列与类似的普通文件相比消耗额外的存储空间。

注意点

使用 `Nullable` 几乎总是对性能产生负面影响，在设计数据库时请记住这一点

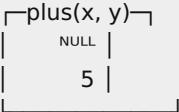
掩码文件中的条目允许 ClickHouse 区分每个表行的对应数据类型的 «NULL» 和默认值由于有额外的文件，«`Nullable`» 列比普通列消耗更多的存储空间

用法示例

```
CREATE TABLE t_null(x Int8, y Nullable(Int8)) ENGINE TinyLog
```

```
INSERT INTO t_null VALUES (1, NULL), (2, 3)
```

```
SELECT x + y FROM t_null
```



固定字符串

固定长度 N 的字符串 (N 必须是严格的正自然数)。

您可以使用下面的语法对列声明为 `FixedString` 类型：

```
<column_name> FixedString(N)
```

其中 N 表示自然数。

当数据的长度恰好为 N 个字节时，`FixedString` 类型是高效的。在其他情况下，这可能会降低效率。

可以有效存储在 `FixedString` 类型的列中的值的示例：

- 二进制表示的IP地址（IPv6使用 `FixedString(16)`）
- 语言代码（ru_RU, en_US ...）
- 货币代码（USD, RUB ...）
- 二进制表示的哈希值（MD5使用 `FixedString(16)`，SHA256使用 `FixedString(32)`）

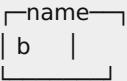
请使用 `UUID` 数据类型来存储 `UUID` 值，。

当向 ClickHouse 中插入数据时，

- 如果字符串包含的字节数少于 ' N '，将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 ，将抛出 `Too large value for FixedString(N)` 异常。

当做数据查询时，ClickHouse 不会删除字符串末尾的空字节。如果使用 `WHERE` 子句，则须要手动添加空字节以匹配 `FixedString` 的值。以下示例阐明了如何将 `WHERE` 子句与 `FixedString` 一起使用。

考虑带有 `FixedString(2)` 列的表：



查询语句 `SELECT * FROM FixedStringTable WHERE a = 'b'` 不会返回任何结果。请使用空字节来填充筛选条件。

```
SELECT * FROM FixedStringTable  
WHERE a = 'b\0'
```



这种方式与 MySQL 的 `CHAR` 类型的方式不同（MySQL 中使用空格填充字符串，并在输出时删除空格）。

请注意，`FixedString(N)` 的长度是个常量。仅由空字符组成的字符串，函数 `length` 返回值为 N ，而函数 `empty` 的返回值为 1。

字符串

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。因此，字符串类型可以代替其他 DBMSs 中的 VARCHAR、BLOB、CLOB 等类型。

编码

ClickHouse 没有编码的概念。字符串可以是任意的字节集，按它们原本的方式进行存储和输出。

若需存储文本，我们建议使用 UTF-8 编码。至少，如果你的终端使用 UTF-8（推荐），这样读写就不需要进行任何的转换了。

同样，对不同的编码文本 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，然而 `lengthUTF8` 函数是假设字符串以 UTF-8 编码，计算的是字符串包含的 Unicode 字符的长度。

布尔值

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。

数据类型

ClickHouse 可以在数据表中存储多种数据类型。

本节描述 ClickHouse 支持的数据类型，以及使用或者实现它们时（如果有的话）的注意事项。

日期

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。允许存储从 Unix 纪元开始到编译阶段定义的上限阈值常量（目前上限是 2106 年，但最终完全支持的年份为 2105）。最小值输出为 0000-00-00。

日期中没有存储时区信息。

日期时间

时间戳类型。用四个字节（无符号的）存储 Unix 时间戳）。允许存储与日期类型相同的范围内的值。最小值为 0000-00-00 00:00:00。时间戳类型值精确到秒（没有闰秒）。

时区

使用启动客户端或服务器时的系统时区，时间戳是从文本（分解为组件）转换为二进制并返回。在文本格式中，有关夏令时的信息会丢失。

默认情况下，客户端连接到服务的时候会使用服务端时区。您可以通过启用客户端命令行选项 `--use_client_time_zone` 来设置使用客户端时间。

因此，在处理文本日期时（例如，在保存文本转储时），请记住在夏令时更改期间可能存在歧义，如果时区发生更改，则可能存在匹配数据的问题。

阵列(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，不能存储在 `MergeTree` 表中存储多维数组。

创建数组

您可以使用 `array` 函数来创建数组：

```
array(T)
```

您也可以使用方括号：

```
[]
```

创建数组示例：

```
:) 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.
```

使用数据类型

ClickHouse会自动检测数组元素，并根据元素计算出存储这些元素最小的数据类型。如果在元素中存在 **NULL** 或存在 **可为空** 类型元素，那么数组的元素类型将会变成 **可为空**。

如果 ClickHouse 无法确定数据类型，它将产生异常。当尝试同时创建一个包含字符串和数字的数组时会发生这种情况 (`SELECT array(1, 'a')`)。

自动数据类型检测示例：

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

```
SELECT
```

```
    [1, 2, NULL] AS x,  
    toTypeName(x)
```

```
└── x ── toTypeName(array(1, 2, NULL)) ─  
   | [1,2,NULL] | Array(Nullable(UInt8)) |
```

```
1 rows in set. Elapsed: 0.002 sec.
```

如果您尝试创建不兼容的数据类型数组，ClickHouse 将引发异常：

```
:) SELECT array(1, 'a')
```

```
SELECT [1, 'a']
```

```
Received exception from server (version 1.1.54388):
```

```
Caused by: 206: DB::Exception: Data type mismatch: cannot convert string to UInt8 at line 1 column 1 (select array(1, 'a'))
```

```
Code: 386. DB::Exception: Received from localhost:9000, 127.0.0.1. DB::Exception: There is no supertype for types UInt8, String because some of them are String/FixedString and some of them are not.
```

```
0 rows in set. Elapsed: 0.246 sec.
```

Nested(Name1 Type1, Name2 Type2, ...)

嵌套数据结构类似于嵌套表。嵌套数据结构的参数（列名和类型）与 CREATE 查询类似。每个表可以包含任意多行嵌套数据结构。

示例：

```
CREATE TABLE test.visits
(
    CounterID UInt32,
    StartDate Date,
    Sign Int8,
    IsNew UInt8,
    VisitID UInt64,
    UserID UInt64,
    ...
    Goals Nested
    (
        ID UInt32,
        Serial UInt32,
        EventTime DateTime,
        Price Int64,
        OrderID String,
        CurrencyID UInt32
    ),
    ...
) ENGINE = CollapsingMergeTree(StartDate, intHash32(UserID), (CounterID, StartDate, intHash32(UserID), VisitID), 8192,
Sign)
```

上述示例声明了 Goals 这种嵌套数据结构，它包含访客转化相关的数据（访客达到的目标）。在 ‘visits’ 表中每一行都可以对应零个或者任意个转化数据。

只支持一级嵌套。嵌套结构的列中，若列的类型是数组类型，那么该列其实和多维数组是相同的，所以目前嵌套层级的支持很局限（MergeTree 引擎中不支持存储这样的列）

大多数情况下，处理嵌套数据结构时，会指定一个单独的列。为了这样实现，列的名称会与点号连接起来。这些列构成了一组匹配类型。在同一条嵌套数据中，所有的列都具有相同的长度。

示例：

```
SELECT
    Goals.ID,
    Goals.EventTime
FROM test.visits
WHERE CounterID = 101500 AND length(Goals.ID) < 5
LIMIT 10
```

Goals.ID	Goals.EventTime
[1073752, 591325, 591325]	['2014-03-17 16:38:10', '2014-03-17 16:38:48', '2014-03-17 16:42:27']
[1073752]	['2014-03-17 00:28:25']
[1073752]	['2014-03-17 10:46:20']
[1073752, 591325, 591325, 591325]	['2014-03-17 13:59:20', '2014-03-17 22:17:55', '2014-03-17 22:18:07', '2014-03-17 22:18:07']

```

1/ 22:18:51' | [ ] | [ ]
| [1073752,591325,591325] | ['2014-03-17 11:37:06','2014-03-17 14:07:47','2014-03-17 14:36:21'] |
| [ ] | [ ] | [ ]
| [ ] | [ ] | [ ]
| [591325,1073752] | ['2014-03-17 00:46:05','2014-03-17 00:46:05'] |
| [1073752,591325,591325,591325] | ['2014-03-17 13:28:33','2014-03-17 13:30:26','2014-03-17 18:51:21','2014-03-17 18:51:45'] |

```

所以可以简单地把嵌套数据结构当做是所有列都是相同长度的多列数组。

SELECT 查询只有在使用 **ARRAY JOIN** 的时候才可以指定整个嵌套数据结构的名称。更多信息，参考 «**ARRAY JOIN** 子句»。示例：

```

SELECT
    Goal.ID,
    Goal.EventTime
FROM test.visits
ARRAY JOIN Goals AS Goal
WHERE CounterID = 101500 AND length(Goals.ID) < 5
LIMIT 10

```

Goal.ID	Goal.EventTime
1073752	2014-03-17 16:38:10
591325	2014-03-17 16:38:48
591325	2014-03-17 16:42:27
1073752	2014-03-17 00:28:25
1073752	2014-03-17 10:46:20
1073752	2014-03-17 13:59:20
591325	2014-03-17 22:17:55
591325	2014-03-17 22:18:07
591325	2014-03-17 22:18:51
1073752	2014-03-17 11:37:06

不能对整个嵌套数据结构执行 **SELECT**。只能明确列出属于它一部分列。

对于 **INSERT** 查询，可以单独地传入所有嵌套数据结构中的列数组（假如它们是单独的列数组）。在插入过程中，系统会检查它们是否有相同的长度。

对于 **DESCRIBE** 查询，嵌套数据结构中的列会以相同的方式分别列出来。

ALTER 查询对嵌套数据结构的操作非常有限。

嵌套数据结构

间隔

表示时间和日期间隔的数据类型族。由此产生的类型 **INTERVAL** 接线员

警告

Interval 数据类型值不能存储在表中。

结构:

- 时间间隔作为无符号整数值。
- 间隔的类型。

支持的时间间隔类型:

- SECOND
- MINUTE
- HOUR
- DAY
- WEEK
- MONTH
- QUARTER
- YEAR

对于每个间隔类型，都有一个单独的数据类型。例如，DAY 间隔对应于 IntervalDay 数据类型:

```
SELECT toTypeName(INTERVAL 4 DAY)
```

```
└─toTypeName(toIntervalDay(4))─  
| IntervalDay |
```

使用说明

您可以使用 Interval - 在算术运算类型值 日期 和 日期时间 - 类型值。例如，您可以将 4 天添加到当前时间:

```
SELECT now() as current_date_time, current_date_time + INTERVAL 4 DAY
```

```
└─current_date_time─ plus(now(), toIntervalDay(4))─  
| 2019-10-23 10:58:45 | 2019-10-27 10:58:45 |
```

不同类型的间隔不能合并。你不能使用间隔，如 4 DAY 1 HOUR. 以小于或等于间隔的最小单位指定间隔，例如，间隔 1 day and an hour 间隔可以表示为 25 HOUR 或 90000 SECOND.

你不能执行算术运算 Interval - 类型值，但你可以添加不同类型的时间间隔，因此值 Date 或 DateTime 数据类型。例如:

```
SELECT now() AS current_date_time, current_date_time + INTERVAL 4 DAY + INTERVAL 3 HOUR
```

```
└─current_date_time─ plus(plus(now(), toIntervalDay(4)), toIntervalHour(3))─  
| 2019-10-23 11:16:28 | 2019-10-27 14:16:28 |
```

以下查询将导致异常:

```
select now() AS current_date_time, current_date_time + (INTERVAL 4 DAY + INTERVAL 3 HOUR)
```

```
Received exception from server (version 19.14.1):
```

Code: 43. DB::Exception: Received from localhost:9000. DB::Exception: Wrong argument types for function plus: if one argument is Interval, then another must be Date or DateTime..

另请参阅

- [INTERVAL](#) 接线员
- [toInterval](#) 类型转换函数

没什么

此数据类型的唯一目的是表示不是期望值的情况。所以不能创建一个 `Nothing` 类型的值。

例如，文本 `NULL` 的类型为 `Nullable(Nothing)`。详情请见 [可为空](#)。

`Nothing` 类型也可以用来表示空数组：

```
:) SELECT toTypeName(array())
SELECT toTypeName([])
└─toTypeName(array())─
  └─Array(Nothing)  ─
1 rows in set. Elapsed: 0.062 sec.
```

特殊数据类型

特殊数据类型的值既不能存在表中也不能在结果中输出，但可用于查询的中间结果。

表达式

用于表示高阶函数中的Lambda表达式。

设置

可以用在 `IN` 表达式的右半部分。

ClickHouse指南

详细的一步一步的说明，帮助解决使用ClickHouse的各种任务列表：

- [简单集群设置教程](#)
- [在ClickHouse中应用CatBoost模型](#)

在ClickHouse中应用Catboost模型

[CatBoost](#) 是一个自由和开源的梯度提升库开发 [Yandex](#) 用于机器学习。

通过此指令，您将学习如何通过从SQL运行模型推理在ClickHouse中应用预先训练好的模型。

在ClickHouse中应用CatBoost模型：

1. [创建表](#).
2. [将数据插入到表中](#).
3. 碰莽祿[into拢Integrate010-68520682<url>](#) (可选步骤) .
4. [从SQL 运行模型推理](#).

有关训练CatBoost模型的详细信息，请参阅 [培训和应用模型](#)。

先决条件

如果你没有 Docker 然而，安装它。

注

Docker 是一个软件平台，允许您创建容器，将CatBoost和ClickHouse安装与系统的其余部分隔离。

在应用CatBoost模型之前：

1. 拉 码头窗口映像 从注册表：

```
$ docker pull yandex/tutorial-catboost-clickhouse
```

此Docker映像包含运行CatBoost和ClickHouse所需的所有内容：代码、运行时、库、环境变量和配置文件。

2. 确保已成功拉取Docker映像：

```
$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
yandex/tutorial-catboost-clickhouse  latest    622e4d17945b   22 hours ago  1.37GB
```

3. 基于此映像启动一个Docker容器：

```
$ docker run -it -p 8888:8888 yandex/tutorial-catboost-clickhouse
```

1. 创建表

为训练样本创建ClickHouse表：

1. 在交互模式下启动ClickHouse控制台客户端：

```
$ clickhouse client
```

注

ClickHouse服务器已经在Docker容器内运行。

2. 使用以下命令创建表：

```
:) CREATE TABLE amazon_train
(
    date Date MATERIALIZED today(),
    ACTION UInt8,
    RESOURCE UInt32,
    MGR_ID UInt32,
    ROLE_ROLLUP_1 UInt32,
    ROLE_ROLLUP_2 UInt32,
    ROLE_DEPTNAME UInt32,
    ROLE_TITLE UInt32,
    ROLE_FAMILY_DESC UInt32
```

```
ROLE_FAMILY UInt32,
ROLE_FAMILY UInt32,
ROLE_CODE UInt32
)
ENGINE = MergeTree ORDER BY date
```

3. 从ClickHouse控制台客户端退出:

```
:) exit
```

2. 将数据插入到表中

插入数据:

1. 运行以下命令:

```
$ clickhouse client --host 127.0.0.1 --query 'INSERT INTO amazon_train FORMAT CSVWithNames' < ~/amazon/train.csv
```

2. 在交互模式下启动ClickHouse控制台客户端:

```
$ clickhouse client
```

3. 确保数据已上传:

```
:) SELECT count() FROM amazon_train
SELECT count()
FROM amazon_train
+-count()-+
| 65538 |
+-----+
```

3. 碌莽祿into拢Integrate010-68520682\<url>

注

可选步骤。 Docker映像包含运行CatBoost和ClickHouse所需的所有内容。

碌莽祿to拢integrate010-68520682\<url>:

1. 构建评估库。

评估CatBoost模型的最快方法是编译 `libcatboostmodel.<so|dll|dylib>` 图书馆。有关如何构建库的详细信息，请参阅 [CatBoost文件](#)。

2. 例如，在任何地方和任何名称创建一个新目录，`data` 并将创建的库放入其中。 Docker映像已经包含了库 `data/libcatboostmodel.so`。

3. 例如，在任何地方和任何名称为 `config model` 创建一个新目录，`models`。

4. 创建具有任意名称的模型配置文件，例如，`models/amazon_model.xml`。

5. 描述模型配置:

```
<models>
<model>
    <!-- Model type. Now catboost only. -->
    <type>catboost</type>
    <!-- Model name. -->
    <name>amazon</name>
    <!-- Path to trained model. -->
    <path>/home/catboost/tutorial/catboost_model.bin</path>
    <!-- Update interval. -->
    <lifetime>0</lifetime>
</model>
</models>
```

6. 将CatBoost的路径和模型配置添加到ClickHouse配置：

```
<!-- File etc/clickhouse-server/config.d/models_config.xml. -->
<catboost_dynamic_library_path>/home/catboost/data/libcatboostmodel.so</catboost_dynamic_library_path>
<models_config>/home/catboost/models/*_model.xml</models_config>
```

4. 从SQL运行模型推理

对于测试模型，运行ClickHouse客户端 `$ clickhouse client`.

让我们确保模型正常工作：

```
:) SELECT
    modelEvaluate('amazon',
        RESOURCE,
        MGR_ID,
        ROLE_ROLLUP_1,
        ROLE_ROLLUP_2,
        ROLE_DEPTNAME,
        ROLE_TITLE,
        ROLE_FAMILY_DESC,
        ROLE_FAMILY,
        ROLE_CODE) > 0 AS prediction,
    ACTION AS target
FROM amazon_train
LIMIT 10
```

注

功能 `模型值` 返回带有多类模型的每类原始预测的元组。

让我们预测一下：

```
:) SELECT
    modelEvaluate('amazon',
        RESOURCE,
        MGR_ID,
        ROLE_ROLLUP_1,
        ROLE_ROLLUP_2,
        ROLE_DEPTNAME,
        ROLE_TITLE,
        ROLE_FAMILY_DESC,
        ROLE_FAMILY,
        ROLE_CODE) AS prediction,
```

```
1. / (1 + exp(-prediction)) AS probability,  
ACTION AS target  
FROM amazon_train  
LIMIT 10
```

注

更多信息 [exp\(\)](#) 功能。

让我们计算样本的LogLoss:

```
:) SELECT -avg(tg * log(prob) + (1 - tg) * log(1 - prob)) AS logloss  
FROM  
(  
    SELECT  
        modelEvaluate('amazon',  
            RESOURCE,  
            MGR_ID,  
            ROLE_ROLLUP_1,  
            ROLE_ROLLUP_2,  
            ROLE_DEPTNAME,  
            ROLE_TITLE,  
            ROLE_FAMILY_DESC,  
            ROLE_FAMILY,  
            ROLE_CODE) AS prediction,  
        1. / (1. + exp(-prediction)) AS prob,  
        ACTION AS tg  
    FROM amazon_train  
)
```

注

更多信息 [avg\(\)](#) 和 [日志\(\)](#) 功能。

ツ环板 Providers ヨツ嘉ツ

信息

如果您已经启动了带有托管ClickHouse服务的公共云，请随时 [打开拉取请求](#) 将其添加到以下列表。

Yandex云

Yandex的ClickHouse托管服务 提供以下主要功能:

- 全面管理的动物园管理员服务 ClickHouse 复制
- 多种存储类型选择
- 不同可用区中的副本
- 加密和隔离
- 自动化维护

ClickHouse Commercial Support Service Providers

Info

If you have launched a ClickHouse commercial support service, feel free to [open a pull-request](#) adding it to the following list.

Altinity

Altinity has offered enterprise ClickHouse support and services since 2017. Altinity customers range from Fortune 100 enterprises to startups. Visit www.altinity.com for more information.

Mafiree

[Service description](#)

MinervaDB

[Service description](#)

ClickHouse release v20.4

ClickHouse release v20.4.2.9, 2020-05-12

Backward Incompatible Change

- System tables (e.g. system.query_log, system.trace_log, system.metric_log) are using compact data part format for parts smaller than 10 MiB in size. Compact data part format is supported since version 20.3. If you are going to downgrade to version less than 20.3, you should manually delete table data for system logs in `/var/lib/clickhouse/data/system/`.
- When string comparison involves FixedString and compared arguments are of different sizes, do comparison as if smaller string is padded to the length of the larger. This is intended for SQL compatibility if we imagine that FixedString data type corresponds to SQL CHAR. This closes [#9272](#). [#10363 \(alexey-milovidov\)](#)
- Make SHOW CREATE TABLE multiline. Now it is more readable and more like MySQL. [#10049 \(Azat Khuzhin\)](#)
- Added a setting validate_polygons that is used in `pointInPolygon` function and enabled by default. [#9857 \(alexey-milovidov\)](#)

New Feature

- Add support for secured connection from ClickHouse to Zookeeper [#10184 \(Konstantin Lebedev\)](#)
- Support custom HTTP handlers. See ISSUES-5436 for description. [#7572 \(Winter Zhang\)](#)
- Add MessagePack Input/Output format. [#9889 \(Kruglov Pavel\)](#)
- Add Regexp input format. [#9196 \(Kruglov Pavel\)](#)
- Added output format `Markdown` for embedding tables in markdown documents. [#10317 \(Kruglov Pavel\)](#)
- Added support for custom settings section in dictionaries. Also fixes issue [#2829](#). [#10137 \(Artem Streltsov\)](#)
- Added custom settings support in DDL-queries for CREATE DICTIONARY [#10465 \(Artem Streltsov\)](#)
- Add simple server-wide memory profiler that will collect allocation contexts when server memory usage becomes higher than the next allocation threshold. [#10444 \(alexey-milovidov\)](#)
- Add setting `always_fetch_merged_part` which restrict replica to merge parts by itself and always prefer downloading from other replicas. [#10379 \(alesapin\)](#)

- Add function `JSONExtractKeysAndValuesRaw` which extracts raw data from JSON objects #10378 (hczi)
- Add memory usage from OS to `system.asynchronous_metrics`. #10361 (alexey-milovidov)
- Added generic variants for functions `least` and `greatest`. Now they work with arbitrary number of arguments of arbitrary types. This fixes #4767 #10318 (alexey-milovidov)
- Now ClickHouse controls timeouts of dictionary sources on its side. Two new settings added to cache dictionary configuration: `strict_max_lifetime_seconds`, which is `max_lifetime` by default, and `query_timeout_milliseconds`, which is one minute by default. The first setting is also useful with `allow_read_expired_keys` settings (to forbid reading very expired keys). #10337 (Nikita Mikhaylov)
- Add `log_queries_min_type` to filter which entries will be written to `query_log` #10053 (Azat Khuzhin)
- Added function `isConstant`. This function checks whether its argument is constant expression and returns 1 or 0. It is intended for development, debugging and demonstration purposes. #10198 (alexey-milovidov)
- add `joinGetOrNull` to return NULL when key is missing instead of returning the default value. #10094 (Amos Bird)
- Consider `NULL` to be equal to `NULL` in `IN` operator, if the option `transform_null_in` is set. #10085 (achimbab)
- Add `ALTER TABLE ... RENAME COLUMN` for MergeTree table engines family. #9948 (alesapin)
- Support parallel distributed `INSERT SELECT`. #9759 (vxider)
- Add ability to query Distributed over Distributed (w/o `distributed_group_by_no_merge`) ... #9923 (Azat Khuzhin)
- Add function `arrayReduceInRanges` which aggregates array elements in given ranges. #9598 (hczi)
- Add Dictionary Status on prometheus exporter. #9622 (Guillaume Tassery)
- Add function `arrayAUC` #8698 (taiyang-li)
- Support `DROP VIEW` statement for better TPC-H compatibility. #9831 (Amos Bird)
- Add 'strict_order' option to `windowFunnel()` #9773 (achimbab)
- Support `DATE` and `TIMESTAMP` SQL operators, e.g. `SELECT date '2001-01-01'` #9691 (Artem Zuikov)

Experimental Feature

- Added experimental database engine Atomic. It supports non-blocking `DROP` and `RENAME TABLE` queries and atomic `EXCHANGE TABLES t1 AND t2` query #7512 (tavplubix)
- Initial support for ReplicatedMergeTree over S3 (it works in suboptimal way) #10126 (Pavel Kovalenko)

Bug Fix

- Fixed incorrect scalar results inside inner query of `MATERIALIZED VIEW` in case if this query contained dependent table #10603 (Nikolai Kochetov)
- Fixed bug, which caused HTTP requests to get stuck on client closing connection when `readonly=2` and `cancel_http_readonly_queries_on_client_close=1`. #10684 (tavplubix)
- Fix segfault in `StorageBuffer` when exception is thrown on server startup. Fixes #10550 #10609 (tavplubix)
- The query `SYSTEM DROP DNS CACHE` now also drops caches used to check if user is allowed to connect from some IP addresses #10608 (tavplubix)
- Fix usage of multiple `IN` operators with an identical set in one query. Fixes #10539 #10686 (Anton Popov)
- Fix crash in `generateRandom` with nested types. Fixes #10583. #10734 (Nikolai Kochetov)
- Fix data corruption for `LowCardinality(FixedString)` key column in `SummingMergeTree` which could have happened after merge. Fixes #10489. #10721 (Nikolai Kochetov)
- Fix logic for `aggregation_memory_efficient_merge_threads` setting. #10667 (palasonic1)
- Fix disappearing totals. Totals could have been filtered if query had `JOIN` or subquery with external `WHERE` condition. Fixes #10674 #10698 (Nikolai Kochetov)
- Fix the lack of parallel execution of remote queries with `distributed_aggregation_memory_efficient` enabled. Fixes #10655 #10664 (Nikolai Kochetov)
- Fix possible incorrect number of rows for queries with `LIMIT`. Fixes #10566, #10709 #10660 (Nikolai Kochetov)
- Fix index corruption, which may occur in some cases after merging compact parts into another compact

part. #10531 ([Anton Popov](#))

- Fix the situation, when mutation finished all parts, but hung up in `is_done=0`. #10526 ([alesapin](#))
- Fix overflow at beginning of unix epoch for timezones with fractional offset from UTC. Fixes #9335. #10513 ([alexey-milovidov](#))
- Better diagnostics for input formats. Fixes #10204 #10418 ([tavplubix](#))
- Fix numeric overflow in `simpleLinearRegression()` over large integers #10474 ([hc2](#))
- Fix use-after-free in Distributed shutdown, avoid waiting for sending all batches #10491 ([Azat Khuzhin](#))
- Add CA certificates to clickhouse-server docker image #10476 ([filimonov](#))
- Fix a rare endless loop that might have occurred when using the `addressToLine` function or `AggregateFunctionState` columns. #10466 ([Alexander Kuzmenkov](#))
- Handle zookeeper "no node error" during distributed query #10050 ([Daniel Chen](#))
- Fix bug when server cannot attach table after column's default was altered. #10441 ([alesapin](#))
- Implicitly cast the default expression type to the column type for the ALIAS columns #10563 ([Azat Khuzhin](#))
- Don't remove metadata directory if `ATTACH DATABASE` fails #10442 ([Winter Zhang](#))
- Avoid dependency on system tzdata. Fixes loading of `Africa/Casablanca` timezone on CentOS 8. Fixes #10211 #10425 ([alexey-milovidov](#))
- Fix some issues if data is inserted with quorum and then gets deleted (DROP PARTITION, TTL, etc.). It led to stuck of INSERTs or false-positive exceptions in SELECTs. Fixes #9946 #10188 ([Nikita Mikhaylov](#))
- Check the number and type of arguments when creating BloomFilter index #9623 #10431 ([Winter Zhang](#))
- Prefer `fallback_to_stale_replicas` over `skip_unavailable_shards`, otherwise when both settings specified and there are no up-to-date replicas the query will fail (patch from @alex-zaitsev) #10422 ([Azat Khuzhin](#))
- Fix the issue when a query with ARRAY JOIN, ORDER BY and LIMIT may return incomplete result. Fixes #10226. #10427 ([Vadim Plakhtinskiy](#))
- Add database name to dictionary name after DETACH/ATTACH. Fixes `system.dictionaries` table and `SYSTEM RELOAD` query #10415 ([Azat Khuzhin](#))
- Fix possible incorrect result for extremes in processors pipeline. #10131 ([Nikolai Kochetov](#))
- Fix possible segfault when the setting `distributed_group_by_no_merge` is enabled (introduced in 20.3.7.46 by #10131). #10399 ([Nikolai Kochetov](#))
- Fix wrong flattening of `Array(Tuple(...))` data types. Fixes #10259 #10390 ([alexey-milovidov](#))
- Fix column names of constants inside JOIN that may clash with names of constants outside of JOIN #9950 ([Alexander Kuzmenkov](#))
- Fix order of columns after `Block::sortColumns()` #10826 ([Azat Khuzhin](#))
- Fix possible `Pipeline` stuck error in `ConcatProcessor` which may happen in remote query. #10381 ([Nikolai Kochetov](#))
- Don't make disk reservations for aggregations. Fixes #9241 #10375 ([Azat Khuzhin](#))
- Fix wrong behaviour of datetime functions for timezones that has altered between positive and negative offsets from UTC (e.g. Pacific/Kiritimati). Fixes #7202 #10369 ([alexey-milovidov](#))
- Avoid infinite loop in `dictIsIn` function. Fixes #515 #10365 ([alexey-milovidov](#))
- Disable GROUP BY sharding_key optimization by default and fix it for WITH ROLLUP/CUBE/TOTALS #10516 ([Azat Khuzhin](#))
- Check for error code when checking parts and don't mark part as broken if the error is like "not enough memory". Fixes #6269 #10364 ([alexey-milovidov](#))
- Show information about not loaded dictionaries in system tables. #10234 ([Vitaly Baranov](#))
- Fix nullptr dereference in `StorageBuffer` if server was shutdown before table startup. #10641 ([alexey-milovidov](#))
- Fixed `DROP` vs `OPTIMIZE` race in `ReplicatedMergeTree`. `DROP` could leave some garbage in replica path in ZooKeeper if there was concurrent `OPTIMIZE` query. #10312 ([tavplubix](#))
- Fix 'Logical error: CROSS JOIN has expressions' error for queries with comma and names joins mix. Fixes #9910 #10311 ([Artem Zuikov](#))
- Fix queries with `max_bytes_before_external_group_by`. #10302 ([Artem Zuikov](#))
- Fix the issue with limiting maximum recursion depth in parser in certain cases. This fixes #10283. This fix may introduce minor incompatibility: long and deep queries via clickhouse-client may refuse to work, and you should adjust settings `max_query_size` and `max_parser_depth` accordingly. #10295 ([alexey-milovidov](#))

milovidov)

- Allow to use `count(*)` with multiple JOINS. Fixes #9853 #10291 (Artem Zuikov)
- Fix error Pipeline stuck with `max_rows_to_group_by` and `group_by_overflow_mode = 'break'`. #10279 (Nikolai Kochetov)
- Fix 'Cannot add column' error while creating `range_hashed` dictionary using DDL query. Fixes #10093. #10235 (alesapin)
- Fix rare possible exception `Cannot drain connections: cancel first.` #10239 (Nikolai Kochetov)
- Fixed bug where ClickHouse would throw "Unknown function lambda." error message when user tries to run ALTER UPDATE/DELETE on tables with ENGINE = Replicated*. Check for nondeterministic functions now handles lambda expressions correctly. #10237 (Alexander Kazakov)
- Fixed reasonably rare segfault in StorageSystemTables that happens when SELECT ... FROM system.tables is run on a database with Lazy engine. #10209 (Alexander Kazakov)
- Fix possible infinite query execution when the query actually should stop on LIMIT, while reading from infinite source like `system.numbers` or `system.zeros`. #10206 (Nikolai Kochetov)
- Fixed "generateRandom" function for Date type. This fixes #9973. Fix an edge case when dates with year 2106 are inserted to MergeTree tables with old-style partitioning but partitions are named with year 1970. #10218 (alexey-milovidov)
- Convert types if the table definition of a View does not correspond to the SELECT query. This fixes #10180 and #10022 #10217 (alexey-milovidov)
- Fix `parseDateTimeBestEffort` for strings in RFC-2822 when day of week is Tuesday or Thursday. This fixes #10082 #10214 (alexey-milovidov)
- Fix column names of constants inside JOIN that may clash with names of constants outside of JOIN. #10207 (alexey-milovidov)
- Fix move-to-prewhere optimization in presence of arrayJoin functions (in certain cases). This fixes #10092 #10195 (alexey-milovidov)
- Fix issue with separator appearing in SCRAMBLE for native mysql-connector-java (JDBC) #10140 (BohuTANG)
- Fix using the current database for an access checking when the database isn't specified. #10192 (Vitaly Baranov)
- Fix ALTER of tables with compact parts. #10130 (Anton Popov)
- Add the ability to relax the restriction on non-deterministic functions usage in mutations with `allow_nondeterministic_mutations` setting. #10186 (filimonov)
- Fix `DROP TABLE` invoked for dictionary #10165 (Azat Khuzhin)
- Convert blocks if structure does not match when doing `INSERT` into Distributed table #10135 (Azat Khuzhin)
- The number of rows was logged incorrectly (as sum across all parts) when inserted block is split by parts with partition key. #10138 (alexey-milovidov)
- Add some arguments check and support identifier arguments for MySQL Database Engine #10077 (Winter Zhang)
- Fix incorrect `index_granularity_bytes` check while creating new replica. Fixes #10098. #10121 (alesapin)
- Fix bug in `CHECK TABLE` query when table contain skip indices. #10068 (alesapin)
- Fix Distributed-over-Distributed with the only one shard in a nested table #9997 (Azat Khuzhin)
- Fix possible rows loss for queries with `JOIN` and `UNION ALL`. Fixes #9826, #10113. ... #10099 (Nikolai Kochetov)
- Fix bug in dictionary when local clickhouse server is used as source. It may caused memory corruption if types in dictionary and source are not compatible. #10071 (alesapin)
- Fixed replicated tables startup when updating from an old ClickHouse version where `/table(replicas/relica_name/metadata` node doesn't exist. Fixes #10037. #10095 (alesapin)
- Fix error `Cannot clone block with columns because block has 0 columns ... While executing GroupingAggregatedTransform`. It happened when setting `distributed_aggregation_memory_efficient` was enabled, and distributed query read aggregating data with mixed single and two-level aggregation from different shards. #10063 (Nikolai Kochetov)
- Fix deadlock when database with materialized view failed attach at start #10054 (Azat Khuzhin)
- Fix a segmentation fault that could occur in GROUP BY over string keys containing trailing zero bytes

(#8636, #8925). ... #10025 (Alexander Kuzmenkov)

- Fix wrong results of distributed queries when alias could override qualified column name. Fixes #9672 #9714 #9972 (Artem Zuikov)
- Fix possible deadlock in SYSTEM RESTART REPLICAS #9955 (tavplubix)
- Fix the number of threads used for remote query execution (performance regression, since 20.3). This happened when query from `Distributed` table was executed simultaneously on local and remote shards. Fixes #9965 #9971 (Nikolai Kochetov)
- Fixed `DeleteOnDestroy` logic in `ATTACH PART` which could lead to automatic removal of attached part and added few tests #9410 (Vladimir Chebotarev)
- Fix a bug with `ON CLUSTER` DDL queries freezing on server startup. #9927 (Gagan Arneja)
- Fix bug in which the necessary tables weren't retrieved at one of the processing stages of queries to some databases. Fixes #9699. #9949 (achulkov2)
- Fix 'Not found column in block' error when `JOIN` appears with `TOTALS`. Fixes #9839 #9939 (Artem Zuikov)
- Fix parsing multiple hosts set in the `CREATE USER` command #9924 (Vitaly Baranov)
- Fix `TRUNCATE` for Join table engine (#9917). #9920 (Amos Bird)
- Fix race condition between drop and optimize in `ReplicatedMergeTree`. #9901 (alesapin)
- Fix `DISTINCT` for `Distributed` when `optimize_skip_unused_shards` is set. #9808 (Azat Khuzhin)
- Fix "scalar doesn't exist" error in `ALTERs` (#9878). ... #9904 (Amos Bird)
- Fix error with qualified names in `distributed_product_mode='local'`. Fixes #4756 #9891 (Artem Zuikov)
- For `INSERT` queries shards now do clamp the settings from the initiator to their constraints instead of throwing an exception. This fix allows to send `INSERT` queries to a shard with another constraints. This change improves fix #9447. #9852 (Vitaly Baranov)
- Add some retries when committing offsets to Kafka broker, since it can reject commit if during `offsets.commit.timeout.ms` there were no enough replicas available for the `_consumer_offsets` topic #9884 (filimonov)
- Fix `Distributed` engine behavior when virtual columns of the underlying table used in `WHERE` #9847 (Azat Khuzhin)
- Fixed some cases when timezone of the function argument wasn't used properly. #9574 (Vasily Nemkov)
- Fix 'Different expressions with the same alias' error when query has `PREWHERE` and `WHERE` on distributed table and `SET distributed_product_mode = 'local'`. #9871 (Artem Zuikov)
- Fix mutations excessive memory consumption for tables with a composite primary key. This fixes #9850. #9860 (alesapin)
- Fix calculating grants for introspection functions from the setting `allow_introspection_functions`. #9840 (Vitaly Baranov)
- Fix `max_distributed_connections` (w/ and w/o Processors) #9673 (Azat Khuzhin)
- Fix possible exception Got 0 in totals chunk, expected 1 on client. It happened for queries with `JOIN` in case if right joined table had zero rows. Example: `select * from system.one t1 join system.one t2 on t1(dummy = t2(dummy limit 0 FORMAT TabSeparated);`. Fixes #9777. ... #9823 (Nikolai Kochetov)
- Fix 'COMMA to CROSS JOIN rewriter is not enabled or cannot rewrite query' error in case of subqueries with COMMA JOIN out of tables lists (i.e. in `WHERE`). Fixes #9782 #9830 (Artem Zuikov)
- Fix server crashing when `optimize_skip_unused_shards` is set and expression for key can't be converted to its field type #9804 (Azat Khuzhin)
- Fix empty string handling in `splitByString`. #9767 (hcza)
- Fix broken `ALTER TABLE DELETE COLUMN` query for compact parts. #9779 (alesapin)
- Fixed missing `rows_before_limit_at_least` for queries over http (with processors pipeline). Fixes #9730 #9757 (Nikolai Kochetov)
- Fix excessive memory consumption in `ALTER` queries (mutations). This fixes #9533 and #9670. #9754 (alesapin)
- Fix possible permanent "Cannot schedule a task" error. #9154 (Azat Khuzhin)
- Fix bug in backquoting in external dictionaries DDL. Fixes #9619. #9734 (alesapin)
- Fixed data race in `text_log`. It does not correspond to any real bug. #9726 (alexey-milovidov)
- Fix bug in a replication that doesn't allow replication to work if the user has executed mutations on the previous version. This fixes #9645. #9652 (alesapin)

- Fixed incorrect internal function names for `sumKahan` and `sumWithOverflow`. It led to exception while using this functions in remote queries. #9636 (Azat Khuzhin)
- Add setting `use_compact_format_in_distributed_parts_names` which allows to write files for `INSERT` queries into `Distributed` table with more compact format. This fixes #9647. #9653 (alesapin)
- Fix `RIGHT` and `FULL JOIN` with LowCardinality in `JOIN` keys. #9610 (Artem Zuikov)
- Fix possible exceptions `Size of filter doesn't match size of column and Invalid number of rows in Chunk` in `MergeTreeRangeReader`. They could appear while executing `PREWHERE` in some cases. #9612 (Anton Popov)
- Allow `ALTER ON CLUSTER` of `Distributed` tables with internal replication. This fixes #3268 #9617 (shinoi2)
- Fix issue when timezone was not preserved if you write a simple arithmetic expression like `time + 1` (in contrast to an expression like `time + INTERVAL 1 SECOND`). This fixes #5743 #9323 (alexey-milovidov)

Improvement

- Use time zone when comparing `DateTime` with string literal. This fixes #5206. #10515 (alexey-milovidov)
- Print verbose diagnostic info if Decimal value cannot be parsed from text input format. #10205 (alexey-milovidov)
- Add tasks/memory metrics for distributed/buffer schedule pools #10449 (Azat Khuzhin)
- Display result as soon as it's ready for `SELECT DISTINCT` queries in `clickhouse-local` and `HTTP` interface. This fixes #8951 #9559 (alexey-milovidov)
- Allow to use `SAMPLE OFFSET` query instead of `cityHash64(PRIMARY KEY) % N == n` for splitting in `clickhouse-copier`. To use this feature, pass `--experimental-use-sample-offset 1` as a command line argument. #10414 (Nikita Mikhaylov)
- Allow to parse BOM in TSV if the first column cannot contain BOM in its value. This fixes #10301 #10424 (alexey-milovidov)
- Add Avro nested fields insert support #10354 (Andrew Onyshchuk)
- Allowed to alter column in non-modifying data mode when the same type is specified. #10382 (Vladimir Chebotarev)
- Auto `distributed_group_by_no_merge` on `GROUP BY` sharding key (if `optimize_skip_unused_shards` is set) #10341 (Azat Khuzhin)
- Optimize queries with `LIMIT/LIMIT BY/ORDER BY` for distributed with `GROUP BY sharding_key` #10373 (Azat Khuzhin)
- Added a setting `max_server_memory_usage` to limit total memory usage of the server. The metric `MemoryTracking` is now calculated without a drift. The setting `max_memory_usage_for_all_queries` is now obsolete and does nothing. This closes #10293. #10362 (alexey-milovidov)
- Add config option `system_tables_lazy_load`. If it's set to false, then system tables with logs are loaded at the server startup. Alexander Burmak, Svyatoslav Tkhon II Pak, #9642 #10359 (alexey-milovidov)
- Use background thread pool (`background_schedule_pool_size`) for distributed sends #10263 (Azat Khuzhin)
- Use background thread pool for background buffer flushes. #10315 (Azat Khuzhin)
- Support for one special case of removing incompletely written parts. This fixes #9940. #10221 (alexey-milovidov)
- Use `isInjective()` over manual list of such functions for `GROUP BY` optimization. #10342 (Azat Khuzhin)
- Avoid printing error message in log if client sends RST packet immediately on connect. It is typical behaviour of IPVS balancer with keepalived and VRRP. This fixes #1851 #10274 (alexey-milovidov)
- Allow to parse `+inf` for floating point types. This closes #1839 #10272 (alexey-milovidov)
- Implemented `generateRandom` table function for Nested types. This closes #9903 #10219 (alexey-milovidov)
- Provide `max_allowed_packed` in MySQL compatibility interface that will help some clients to communicate with ClickHouse via MySQL protocol. #10199 (BohuTANG)
- Allow literals for GLOBAL IN (i.e. `SELECT * FROM remote('localhost', system.one) WHERE dummy global in (0)`) #10196 (Azat Khuzhin)
- Fix various small issues in interactive mode of `clickhouse-client` #10194 (alexey-milovidov)
- Avoid superfluous dictionaries load (`system.tables`, `DROP/SHOW CREATE TABLE`) #10164 (Azat Khuzhin)

- Update to RWLock: timeout parameter for getLock() + implementation reworked to be phase fair #10073 (Alexander Kazakov)
- Enhanced compatibility with native mysql-connector-java(JDBC) #10021 (BohuTANG)
- The function `toString` is considered monotonic and can be used for index analysis even when applied in tautological cases with String or LowCardinality(String) argument. #10110 (Amos Bird)
- Add `ON CLUSTER` clause support to commands {CREATE|DROP} USER/ROLE/ROW POLICY/SETTINGS PROFILE/QUOTA, GRANT. #9811 (Vitaly Baranov)
- Virtual hosted-style support for S3 URI #9998 (Pavel Kovalenko)
- Now layout type for dictionaries with no arguments can be specified without round brackets in dictionaries DDL-queries. Fixes #10057. #10064 (alesapin)
- Add ability to use number ranges with leading zeros in filepath #9989 (Olga Khvostikova)
- Better memory usage in CROSS JOIN. #10029 (Artem Zuikov)
- Try to connect to all shards in cluster when getting structure of remote table and `skip_unavailable_shards` is set. #7278 (nvartolomei)
- Add `total_rows/total_bytes` into the `system.tables` table. #9919 (Azat Khuzhin)
- System log tables now use polymorphic parts by default. #9905 (Anton Popov)
- Add type column into `system.settings/merge_tree_settings` #9909 (Azat Khuzhin)
- Check for available CPU instructions at server startup as early as possible. #9888 (alexey-milovidov)
- Remove `ORDER BY` stage from mutations because we read from a single ordered part in a single thread. Also add check that the rows in mutation are ordered by sorting key and this order is not violated. #9886 (alesapin)
- Implement operator LIKE for `FixedString` at left hand side. This is needed to better support TPC-DS queries. #9890 (alexey-milovidov)
- Add `force_optimize_skip_unused_shards_no_nested` that will disable `force_optimize_skip_unused_shards` for nested Distributed table #9812 (Azat Khuzhin)
- Now columns size is calculated only once for MergeTree data parts. #9827 (alesapin)
- Evaluate constant expressions for `optimize_skip_unused_shards` (i.e. `SELECT * FROM foo_dist WHERE key=xxHash32(0)`) #8846 (Azat Khuzhin)
- Check for using `Date` or `DateTime` column from TTL expressions was removed. #9967 (Vladimir Chebotarev)
- DiskS3 hard links optimal implementation. #9760 (Pavel Kovalenko)
- If `set multiple_joins_rewriter_version = 2` enables second version of multiple JOIN rewrites that keeps not clashed column names as is. It supports multiple JOINs with `USING` and allow `select *` for JOINs with subqueries. #9739 (Artem Zuikov)
- Implementation of "non-blocking" alter for StorageMergeTree #9606 (alesapin)
- Add MergeTree full support for DiskS3 #9646 (Pavel Kovalenko)
- Extend `splitByString` to support empty strings as separators. #9742 (hc2)
- Add a `timestamp_ns` column to `system.trace_log`. It contains a high-definition timestamp of the trace event, and allows to build timelines of thread profiles ("flame charts"). #9696 (Alexander Kuzmenkov)
- When the setting `send_logs_level` is enabled, avoid intermixing of log messages and query progress. #9634 (Azat Khuzhin)
- Added support of `MATERIALIZE TTL IN PARTITION`. #9581 (Vladimir Chebotarev)
- Support complex types inside Avro nested fields #10502 (Andrew Onyshchuk)

Performance Improvement

- Better insert logic for right table for Partial MergeJoin. #10467 (Artem Zuikov)
- Improved performance of row-oriented formats (more than 10% for CSV and more than 35% for Avro in case of narrow tables). #10503 (Andrew Onyshchuk)
- Improved performance of queries with explicitly defined sets at right side of IN operator and tuples on the left side. #10385 (Anton Popov)
- Use less memory for hash table in HashJoin. #10416 (Artem Zuikov)
- Special HashJoin over StorageDictionary. Allow rewrite `dictGet()` functions with JOINs. It's not backward incompatible itself but could uncover #8400 on some installations. #10133 (Artem Zuikov)
- Enable parallel insert of materialized view when its target table supports. #10052 (vxider)
- Improved performance of index analysis with monotonic functions #9607+10026 (Anton Popov)

- Improved performance of index analysis with monotonic functions. #900 / #10020 (Anton Popov)
- Using SSE2 or SSE4.2 SIMD intrinsics to speed up tokenization in bloom filters. #9968 (Vasily Nemkov)
- Improved performance of queries with explicitly defined sets at right side of `IN` operator. This fixes performance regression in version 20.3. #9740 (Anton Popov)
- Now clickhouse-copier splits each partition in number of pieces and copies them independently. #9075 (Nikita Mikhaylov)
- Adding more aggregation methods. For example TPC-H query 1 will now pick `FixedHashMap<UInt16, AggregateDataPtr>` and gets 25% performance gain #9829 (Amos Bird)
- Use single row counter for multiple streams in pre-limit transform. This helps to avoid uniting pipeline streams in queries with `limit` but without `order by` (like `select f(x) from (select x from t limit 1000000000)`) and use multiple threads for further processing. #9602 (Nikolai Kochetov)

Build/Testing/Packaging Improvement

- Use a fork of AWS SDK libraries from ClickHouse-Extras #10527 (Pavel Kovalenko)
- Add integration tests for new `ALTER RENAME COLUMN` query. #10654 (vzakaznikov)
- Fix possible signed integer overflow in invocation of function `now64` with wrong arguments. This fixes #8973 #10511 (alexey-milovidov)
- Split fuzzer and sanitizer configurations to make build config compatible with Oss-fuzz. #10494 (kyprizel)
- Fixes for clang-tidy on clang-10. #10420 (alexey-milovidov)
- Display absolute paths in error messages. Otherwise KDevelop fails to navigate to correct file and opens a new file instead. #10434 (alexey-milovidov)
- Added `ASAN_OPTIONS` environment variable to investigate errors in CI stress tests with Address sanitizer. #10440 (Nikita Mikhaylov)
- Enable ThinLTO for clang builds (experimental). #10435 (alexey-milovidov)
- Remove accidental dependency on Z3 that may be introduced if the system has Z3 solver installed. #10426 (alexey-milovidov)
- Move integration tests docker files to docker/ directory. #10335 (Ilya Yatsishin)
- Allow to use `clang-10` in CI. It ensures that #10238 is fixed. #10384 (alexey-milovidov)
- Update OpenSSL to upstream master. Fixed the issue when TLS connections may fail with the message `OpenSSL SSL_read: error:14094438:SSL routines:ssl3_read_bytes:tlsv1 alert internal error` and `SSL Exception: error:2400006E:random number generator::error retrieving entropy`. The issue was present in version 20.1. #8956 (alexey-milovidov)
- Fix clang-10 build. <https://github.com/ClickHouse/ClickHouse/issues/10238> #10370 (Amos Bird)
- Add performance test for `Parallel INSERT` for materialized view. #10345 (vxider)
- Fix flaky test `test_settings_constraints_distributed.test_insert_clamps_settings`. #10346 (Vitaly Baranov)
- Add util to test results upload in CI ClickHouse #10330 (Ilya Yatsishin)
- Convert test results to JSONEachRow format in `junit_to_html` tool #10323 (Ilya Yatsishin)
- Update cctz. #10215 (alexey-milovidov)
- Allow to create HTML report from the purest JUnit XML report. #10247 (Ilya Yatsishin)
- Update the check for minimal compiler version. Fix the root cause of the issue #10250 #10256 (alexey-milovidov)
- Initial support for live view tables over distributed #10179 (vzakaznikov)
- Fix (false) MSan report in `MergeTreeIndexFullText`. The issue first appeared in #9968. #10801 (alexey-milovidov)
- clickhouse-docker-util #10151 (filimonov)
- Update pdqsort to recent version #10171 (ivan)
- Update libdivide to v3.0 #10169 (ivan)
- Add check with enabled polymorphic parts. #10086 (Anton Popov)
- Add cross-compile build for FreeBSD. This fixes #9465 #9643 (ivan)
- Add performance test for #6924 #6980 (filimonov)
- Add support of `/dev/null` in the `File` engine for better performance testing #8455 (Amos Bird)
- Move all folders inside `/dbms` one level up #9974 (ivan)
- Add a test that checks that read from `MergeTree` with single thread is performed in order. Addition to #9670 #9762 (alexey-milovidov)

- Fix the `00964_live_view_watch_events_heartbeat.py` test to avoid race condition. #9944 (vzakaznikov)
- Fix integration test `test_settings_constraints` #9962 (Vitaly Baranov)
- Every function in its own file, part 12. #9922 (alexey-milovidov)
- Added performance test for the case of extremely slow analysis of array of tuples. #9872 (alexey-milovidov)
- Update zstd to 1.4.4. It has some minor improvements in performance and compression ratio. If you run replicas with different versions of ClickHouse you may see reasonable error messages `Data after merge is not byte-identical to data on another replicas.` with explanation. These messages are Ok and you should not worry. #10663 (alexey-milovidov)
- Fix TSan report in `system.stack_trace`. #9832 (alexey-milovidov)
- Removed dependency on `clock_getres`. #9833 (alexey-milovidov)
- Added identifier names check with clang-tidy. #9799 (alexey-milovidov)
- Update "builder" docker image. This image is not used in CI but is useful for developers. #9809 (alexey-milovidov)
- Remove old `performance-test` tool that is no longer used in CI. `clickhouse-performance-test` is great but now we are using way superior tool that is doing comparison testing with sophisticated statistical formulas to achieve confident results regardless to various changes in environment. #9796 (alexey-milovidov)
- Added most of clang-static-analyzer checks. #9765 (alexey-milovidov)
- Update Poco to 1.9.3 in preparation for MongoDB URI support. #6892 (Alexander Kuzmenkov)
- Fix build with `-DUSE_STATIC_LIBRARIES=0 -DENABLE_JEMALLOC=0` #9651 (Artem Zuikov)
- For change log script, if merge commit was cherry-picked to release branch, take PR name from commit description. #9708 (Nikolai Kochetov)
- Support `vX.X-conflicts` tag in backport script. #9705 (Nikolai Kochetov)
- Fix `auto-label` for backporting script. #9685 (Nikolai Kochetov)
- Use `libc++` in Darwin cross-build to make it consistent with native build. #9665 (Hui Wang)
- Fix flaky test `01017_uniqCombined_memory_usage`. Continuation of #7236. #9667 (alexey-milovidov)
- Fix build for native MacOS Clang compiler #9649 (Ivan)
- Allow to add various glitches around `pthread_mutex_lock`, `pthread_mutex_unlock` functions. #9635 (alexey-milovidov)
- Add support for `clang-tidy` in `packager` script. #9625 (alexey-milovidov)
- Add ability to use unbundled msgpack. #10168 (Azat Khuzhin)

ClickHouse release v20.3

ClickHouse release v20.3.8.53, 2020-04-23

Bug Fix

- Fixed wrong behaviour of datetime functions for timezones that has altered between positive and negative offsets from UTC (e.g. Pacific/Kiritimati). This fixes #7202 #10369 (alexey-milovidov)
- Fix possible segfault with `distributed_group_by_no_merge` enabled (introduced in 20.3.7.46 by #10131). #10399 (Nikolai Kochetov)
- Fix wrong flattening of `Array(Tuple(...))` data types. This fixes #10259 #10390 (alexey-milovidov)
- Drop disks reservation in Aggregator. This fixes bug in disk space reservation, which may cause big external aggregation to fail even if it could be completed successfully #10375 (Azat Khuzhin)
- Fixed `DROP` vs `OPTIMIZE` race in ReplicatedMergeTree. `DROP` could leave some garbage in replica path in ZooKeeper if there was concurrent `OPTIMIZE` query. #10312 (tavplubix)
- Fix bug when server cannot attach table after column default was altered. #10441 (alesapin)
- Do not remove metadata directory when attach database fails before loading tables. #10442 (Winter Zhang)
- Fixed several bugs when some data was inserted with quorum, then deleted somehow (`DROP PARTITION`, `TTL`) and this led to the stuck of `INSERTs` or false-positive exceptions in `SELECTs`. This fixes #9946 #10188 (Nikita Mikhaylov)
- Fix possible Pipeline stuck error in `ConcatProcessor` which could have happened in remote query. #10381 (Nikolai Kochetov)
- Fixed wrong behavior in `HashTable` that caused compilation error when trying to read `HashMap` from

- Fixed wrong behavior in HashTable that caused compilation error when trying to read Hashmap from buffer. [#10386 \(palasonic1\)](#)
- Allow to use `count(*)` with multiple JOINS. Fixes [#9853 #10291 \(Artem Zuikov\)](#)
- Prefer `fallback_to_stale_replicas` over `skip_unavailable_shards`, otherwise when both settings specified and there are no up-to-date replicas the query will fail (patch from @alex-zaitsev). Fixes: [#2564. #10422 \(Azat Khuzhin\)](#)
- Fix the issue when a query with ARRAY JOIN, ORDER BY and LIMIT may return incomplete result. This fixes [#10226](#). Author: [Vadim Plakhtinskiy](#). [#10427 \(alexey-milovidov\)](#)
- Check the number and type of arguments when creating BloomFilter index [#9623 #10431 \(Winter Zhang\)](#)

Performance Improvement

- Improved performance of queries with explicitly defined sets at right side of `IN` operator and tuples in the left side. This fixes performance regression in version 20.3. [#9740, #10385 \(Anton Popov\)](#)

ClickHouse release v20.3.7.46, 2020-04-17

Bug Fix

- Fix Logical error: CROSS JOIN has expressions error for queries with comma and names joins mix. [#10311 \(Artem Zuikov\)](#).
- Fix queries with `max_bytes_before_external_group_by`. [#10302 \(Artem Zuikov\)](#).
- Fix move-to-prewhere optimization in presence of arrayJoin functions (in certain cases). This fixes [#10092. #10195 \(alexey-milovidov\)](#).
- Add the ability to relax the restriction on non-deterministic functions usage in mutations with `allow_nondeterministic_mutations` setting. [#10186 \(filimonov\)](#).

ClickHouse release v20.3.6.40, 2020-04-16

New Feature

- Added function `isConstant`. This function checks whether its argument is constant expression and returns 1 or 0. It is intended for development, debugging and demonstration purposes. [#10198 \(alexey-milovidov\)](#).

Bug Fix

- Fix error Pipeline stuck with `max_rows_to_group_by` and `group_by_overflow_mode = 'break'`. [#10279 \(Nikolai Kochetov\)](#).
- Fix rare possible exception `Cannot drain connections: cancel first.` [#10239 \(Nikolai Kochetov\)](#).
- Fixed bug where ClickHouse would throw "Unknown function lambda." error message when user tries to run ALTER UPDATE/DELETE on tables with ENGINE = Replicated*. Check for nondeterministic functions now handles lambda expressions correctly. [#10237 \(Alexander Kazakov\)](#).
- Fixed "generateRandom" function for Date type. This fixes [#9973](#). Fix an edge case when dates with year 2106 are inserted to MergeTree tables with old-style partitioning but partitions are named with year 1970. [#10218 \(alexey-milovidov\)](#).
- Convert types if the table definition of a View does not correspond to the SELECT query. This fixes [#10180](#) and [#10022. #10217 \(alexey-milovidov\)](#).
- Fix `parseDateTimeBestEffort` for strings in RFC-2822 when day of week is Tuesday or Thursday. This fixes [#10082. #10214 \(alexey-milovidov\)](#).
- Fix column names of constants inside JOIN that may clash with names of constants outside of JOIN. [#10207 \(alexey-milovidov\)](#).
- Fix possible infinite query execution when the query actually should stop on LIMIT, while reading from infinite source like `system.numbers` or `system.zeros`. [#10206 \(Nikolai Kochetov\)](#).
- Fix using the current database for access checking when the database isn't specified. [#10192 \(Vitaly Baranov\)](#).
- Convert blocks if structure does not match on INSERT into Distributed(). [#10135 \(Azat Khuzhin\)](#).
- Fix possible incorrect result for extremes in processors pipeline. [#10131 \(Nikolai Kochetov\)](#).

- Fix some kinds of alters with compact parts. #10130 (Anton Popov).
- Fix incorrect `index_granularity_bytes` check while creating new replica. Fixes #10098. #10121 (alesapin).
- Fix SIGSEGV on INSERT into Distributed table when its structure differs from the underlying tables. #10105 (Azat Khuzhin).
- Fix possible rows loss for queries with `JOIN` and `UNION ALL`. Fixes #9826, #10113. #10099 (Nikolai Kochetov).
- Fixed replicated tables startup when updating from an old ClickHouse version where `/table/replicas/replica_name/metadata` node doesn't exist. Fixes #10037. #10095 (alesapin).
- Add some arguments check and support identifier arguments for MySQL Database Engine. #10077 (Winter Zhang).
- Fix bug in clickhouse dictionary source from localhost clickhouse server. The bug may lead to memory corruption if types in dictionary and source are not compatible. #10071 (alesapin).
- Fix bug in `CHECK TABLE` query when table contain skip indices. #10068 (alesapin).
- Fix error `Cannot clone block with columns because block has 0 columns ... While executing GroupingAggregatedTransform`. It happened when setting `distributed_aggregation_memory_efficient` was enabled, and distributed query read aggregating data with different level from different shards (mixed single and two level aggregation). #10063 (Nikolai Kochetov).
- Fix a segmentation fault that could occur in GROUP BY over string keys containing trailing zero bytes (#8636, #8925). #10025 (Alexander Kuzmenkov).
- Fix the number of threads used for remote query execution (performance regression, since 20.3). This happened when query from `Distributed` table was executed simultaneously on local and remote shards. Fixes #9965. #9971 (Nikolai Kochetov).
- Fix bug in which the necessary tables weren't retrieved at one of the processing stages of queries to some databases. Fixes #9699. #9949 (achulkov2).
- Fix 'Not found column in block' error when `JOIN` appears with `TOTALS`. Fixes #9839. #9939 (Artem Zuikov).
- Fix a bug with `ON CLUSTER` DDL queries freezing on server startup. #9927 (Gagan Arneja).
- Fix parsing multiple hosts set in the `CREATE USER` command, e.g. `CREATE USER user6 HOST NAME REGEXP 'lo.?*host', NAME REGEXP 'lo**host'`. #9924 (Vitaly Baranov).
- Fix `TRUNCATE` for Join table engine (#9917). #9920 (Amos Bird).
- Fix "scalar doesn't exist" error in ALTERs (#9878). #9904 (Amos Bird).
- Fix race condition between drop and optimize in `ReplicatedMergeTree`. #9901 (alesapin).
- Fix error with qualified names in `distributed_product_mode='local'`. Fixes #4756. #9891 (Artem Zuikov).
- Fix calculating grants for introspection functions from the setting 'allow_introspection_functions'. #9840 (Vitaly Baranov).

Build/Testing/Packaging Improvement

- Fix integration test `test_settings_constraints`. #9962 (Vitaly Baranov).
- Removed dependency on `clock_getres`. #9833 (alexey-milovidov).

ClickHouse release v20.3.5.21, 2020-03-27

Bug Fix

- Fix 'Different expressions with the same alias' error when query has PREWHERE and WHERE on distributed table and `SET distributed_product_mode = 'local'`. #9871 (Artem Zuikov).
- Fix mutations excessive memory consumption for tables with a composite primary key. This fixes #9850. #9860 (alesapin).
- For INSERT queries shard now clamps the settings got from the initiator to the shard's constraints instead of throwing an exception. This fix allows to send INSERT queries to a shard with another constraints. This change improves fix #9447. #9852 (Vitaly Baranov).
- Fix 'COMMA to CROSS JOIN rewriter is not enabled or cannot rewrite query' error in case of subqueries with COMMA JOIN out of tables lists (i.e. in WHERE). Fixes #9782. #9830 (Artem Zuikov).
- Fix possible exception `Got 0 in totals chunk, expected 1 on client`. It happened for queries with `JOIN` in case if right joined table had zero rows. Example: `select * from system.one t1 join system.one t2 on t1.dummy = t2.dummy limit 0 FORMAT TabSeparated`. Fixes #9777. #9823 (Nikolai Kochetov).

`t2.oommy limit 0 FORMAT TabSeparated;. Fixes #9111. #9023 (Nikolai Kochetov).`

- Fix SIGSEGV with `optimize_skip_unused_shards` when type cannot be converted. [#9804](#) (Azat Khuzhin).
- Fix broken `ALTER TABLE DELETE COLUMN` query for compact parts. [#9779](#) (alesapin).
- Fix `max_distributed_connections` (w/ and w/o Processors). [#9673](#) (Azat Khuzhin).
- Fixed a few cases when timezone of the function argument wasn't used properly. [#9574](#) (Vasily Nemkov).

Improvement

- Remove order by stage from mutations because we read from a single ordered part in a single thread. Also add check that the order of rows in mutation is ordered in sorting key order and this order is not violated. [#9886](#) (alesapin).

ClickHouse release v20.3.4.10, 2020-03-20

Bug Fix

- This release also contains all bug fixes from 20.1.8.41
- Fix missing `rows_before_limit_at_least` for queries over http (with processors pipeline). This fixes [#9730](#). [#9757](#) (Nikolai Kochetov)

ClickHouse release v20.3.3.6, 2020-03-17

Bug Fix

- This release also contains all bug fixes from 20.1.7.38
- Fix bug in a replication that doesn't allow replication to work if the user has executed mutations on the previous version. This fixes [#9645](#). [#9652](#) (alesapin). It makes version 20.3 backward compatible again.
- Add setting `use_compact_format_in_distributed_parts_names` which allows to write files for `INSERT` queries into `Distributed` table with more compact format. This fixes [#9647](#). [#9653](#) (alesapin). It makes version 20.3 backward compatible again.

ClickHouse release v20.3.2.1, 2020-03-12

Backward Incompatible Change

- Fixed the issue `file name too long` when sending data for `Distributed` tables for a large number of replicas. Fixed the issue that replica credentials were exposed in the server log. The format of directory name on disk was changed to `[shard{shard_index}][_replica{replica_index}]`. [#8911](#) (Mikhail Korotov) After you upgrade to the new version, you will not be able to downgrade without manual intervention, because old server version does not recognize the new directory format. If you want to downgrade, you have to manually rename the corresponding directories to the old format. This change is relevant only if you have used asynchronous `INSERTs` to `Distributed` tables. In the version 20.3.3 we will introduce a setting that will allow you to enable the new format gradually.
- Changed the format of replication log entries for mutation commands. You have to wait for old mutations to process before installing the new version.
- Implement simple memory profiler that dumps stacktraces to `system.trace_log` every N bytes over soft allocation limit [#8765](#) (Ivan) [#9472](#) (alexey-milovidov) The column of `system.trace_log` was renamed from `timer_type` to `trace_type`. This will require changes in third-party performance analysis and flamegraph processing tools.
- Use OS thread id everywhere instead of internal thread number. This fixes [#7477](#) Old `clickhouse-client` cannot receive logs that are send from the server when the setting `send_logs_level` is enabled, because the names and types of the structured log messages were changed. On the other hand, different server versions can send logs with different types to each other. When you don't use the `send_logs_level` setting, you should not care. [#8954](#) (alexey-milovidov)
- Remove `indexHint` function [#9542](#) (alexey-milovidov)
- Remove `findClusterIndex`, `findClusterValue` functions. This fixes [#8641](#). If you were using these functions, send an email to `clickhouse-feedback@yandex-team.com` [#9543](#) (alexey-milovidov)

- Now it's not allowed to create columns or add columns with `SELECT` subquery as default expression. [#9481 \(alesapin\)](#)
- Require aliases for subqueries in JOIN. [#9274 \(Artem Zuikov\)](#)
- Improved `ALTER MODIFY/ADD` queries logic. Now you cannot `ADD` column without type, `MODIFY` default expression doesn't change type of column and `MODIFY` type doesn't loose default expression value. Fixes [#8669](#). [#9227 \(alesapin\)](#)
- Require server to be restarted to apply the changes in logging configuration. This is a temporary workaround to avoid the bug where the server logs to a deleted log file (see [#8696](#)). [#8707 \(Alexander Kuzmenkov\)](#)
- The setting `experimental_use_processors` is enabled by default. This setting enables usage of the new query pipeline. This is internal refactoring and we expect no visible changes. If you will see any issues, set it to back zero. [#8768 \(alexey-milovidov\)](#)

New Feature

- Add `Avro` and `AvroConfluent` input/output formats [#8571 \(Andrew Onyshchuk\)](#) [#8957 \(Andrew Onyshchuk\)](#) [#8717 \(alexey-milovidov\)](#)
- Multi-threaded and non-blocking updates of expired keys in `cache` dictionaries (with optional permission to read old ones). [#8303 \(Nikita Mikhaylov\)](#)
- Add query `ALTER ... MATERIALIZE TTL`. It runs mutation that forces to remove expired data by TTL and recalculates meta-information about TTL in all parts. [#8775 \(Anton Popov\)](#)
- Switch from HashJoin to MergeJoin (on disk) if needed [#9082 \(Artem Zuikov\)](#)
- Added `MOVE PARTITION` command for `ALTER TABLE` [#4729](#) [#6168 \(Guillaume Tassery\)](#)
- Reloading storage configuration from configuration file on the fly. [#8594 \(Vladimir Chebotarev\)](#)
- Allowed to change `storage_policy` to not less rich one. [#8107 \(Vladimir Chebotarev\)](#)
- Added support for globs/wildcards for S3 storage and table function. [#8851 \(Vladimir Chebotarev\)](#)
- Implement `bitAnd`, `bitOr`, `bitXor`, `bitNot` for `FixedString(N)` datatype. [#9091 \(Guillaume Tassery\)](#)
- Added function `bitCount`. This fixes [#8702](#). [#8708 \(alexey-milovidov\)](#) [#8749 \(ikopylov\)](#)
- Add `generateRandom` table function to generate random rows with given schema. Allows to populate arbitrary test table with data. [#8994 \(Ilya Yatsishin\)](#)
- `JSONEachRowFormat`: support special case when objects enclosed in top-level array. [#8860 \(Kruglov Pavel\)](#)
- Now it's possible to create a column with `DEFAULT` expression which depends on a column with default `ALIAS` expression. [#9489 \(alesapin\)](#)
- Allow to specify `--limit` more than the source data size in `clickhouse-obfuscator`. The data will repeat itself with different random seed. [#9155 \(alexey-milovidov\)](#)
- Added `groupArraySample` function (similar to `groupArray`) with reservoir sampling algorithm. [#8286 \(Amos Bird\)](#)
- Now you can monitor the size of update queue in `cache/complex_key_cache` dictionaries via system metrics. [#9413 \(Nikita Mikhaylov\)](#)
- Allow to use CRLF as a line separator in CSV output format with setting `output_format_csv_crlf_end_of_line` is set to 1 [#8934](#) [#8935](#) [#8963 \(Mikhail Korotov\)](#)
- Implement more functions of the H3 API: `h3GetBaseCell`, `h3HexAreaM2`, `h3IndexesAreNeighbors`, `h3ToChildren`, `h3ToString` and `stringToH3` [#8938 \(Nico Mandery\)](#)
- New setting introduced: `max_parser_depth` to control maximum stack size and allow large complex queries. This fixes [#6681](#) and [#7668](#). [#8647 \(Maxim Smirnov\)](#)
- Add a setting `force_optimize_skip_unused_shards` setting to throw if skipping of unused shards is not possible [#8805 \(Azat Khuzhin\)](#)
- Allow to configure multiple disks/volumes for storing data for send in Distributed engine [#8756 \(Azat Khuzhin\)](#)
- Support storage policy (`<tmp_policy>`) for storing temporary data. [#8750 \(Azat Khuzhin\)](#)
- Added `X-ClickHouse-Exception-Code` HTTP header that is set if exception was thrown before sending data. This implements [#4971](#). [#8786 \(Mikhail Korotov\)](#)
- Added function `ifNotFinite`. It is just a syntactic sugar: `ifNotFinite(x, y) = isFinite(x) ? x : y.` [#8710 \(alexey-milovidov\)](#)
- Added `last_successful_update_time` column in `custom_dictionaries` table. [#8204 \(Nikita Mikhaylov\)](#)

- Added `last_successful_update_time` column in `System.Dictionaries` table #9594 (Nikita Mikhaylov)
- Add `blockSerializedSize` function (size on disk without compression) #8952 (Azat Khuzhin)
- Add function `moduloOrZero` #9358 (hcz)
- Added system tables `system.zeros` and `system.zeros_mt` as well as tale functions `zeros()` and `zeros_mt()`. Tables (and table functions) contain single column with name `zero` and type `UInt8`. This column contains zeros. It is needed for test purposes as the fastest method to generate many rows. This fixes #6604 #9593 (Nikolai Kochetov)

Experimental Feature

- Add new compact format of parts in MergeTree-family tables in which all columns are stored in one file. It helps to increase performance of small and frequent inserts. The old format (one file per column) is now called wide. Data storing format is controlled by settings `min_bytes_for_wide_part` and `min_rows_for_wide_part`. #8290 (Anton Popov)
- Support for S3 storage for `Log`, `TinyLog` and `StripeLog` tables. #8862 (Pavel Kovalenko)

Bug Fix

- Fixed inconsistent whitespaces in log messages. #9322 (alexey-milovidov)
- Fix bug in which arrays of unnamed tuples were flattened as Nested structures on table creation. #8866 (achulkov2)
- Fixed the issue when "Too many open files" error may happen if there are too many files matching glob pattern in `File` table or `file` table function. Now files are opened lazily. This fixes #8857 #8861 (alexey-milovidov)
- DROP TEMPORARY TABLE now drops only temporary table. #8907 (Vitaly Baranov)
- Remove outdated partition when we shutdown the server or DETACH/ATTACH a table. #8602 (Guillaume Tassery)
- For how the default disk calculates the free space from `data` subdirectory. Fixed the issue when the amount of free space is not calculated correctly if the `data` directory is mounted to a separate device (rare case). This fixes #7441 #9257 (Mikhail Korotov)
- Allow comma (cross) join with IN () inside. #9251 (Artem Zuikov)
- Allow to rewrite CROSS to INNER JOIN if there's [NOT] LIKE operator in WHERE section. #9229 (Artem Zuikov)
- Fix possible incorrect result after `GROUP BY` with enabled setting `distributed_aggregation_memory_efficient`. Fixes #9134. #9289 (Nikolai Kochetov)
- Found keys were counted as missed in metrics of cache dictionaries. #9411 (Nikita Mikhaylov)
- Fix replication protocol incompatibility introduced in #8598. #9412 (alesapin)
- Fixed race condition on `queue_task_handle` at the startup of ReplicatedMergeTree tables. #9552 (alexey-milovidov)
- The token `NOT` didn't work in `SHOW TABLES NOT LIKE` query #8727 #8940 (alexey-milovidov)
- Added range check to function `h3EdgeLengthM`. Without this check, buffer overflow is possible. #8945 (alexey-milovidov)
- Fixed up a bug in batched calculations of ternary logical OPs on multiple arguments (more than 10). #8718 (Alexander Kazakov)
- Fix error of PREWHERE optimization, which could lead to segfaults or Inconsistent number of columns got from `MergeTreeRangeReader` exception. #9024 (Anton Popov)
- Fix unexpected `Timeout exceeded while reading from socket` exception, which randomly happens on secure connection before timeout actually exceeded and when query profiler is enabled. Also add `connect_timeout_with_failover_secure_ms` settings (default 100ms), which is similar to `connect_timeout_with_failover_ms`, but is used for secure connections (because SSL handshake is slower, than ordinary TCP connection) #9026 (tavplubix)
- Fix bug with mutations finalization, when mutation may hang in state with `parts_to_do=0` and `is_done=0`. #9022 (alesapin)
- Use new ANY JOIN logic with `partial_merge_join` setting. It's possible to make `ANY|ALL|SEMI LEFT` and `ALL INNER` joins with `partial_merge_join=1` now. #8932 (Artem Zuikov)
- Shard now clamps the settings got from the initiator to the shard's constraints instead of throwing an exception. This fix allows to send queries to a shard with another constraints. #9447 (Vitaly Baranov)

- Fixed memory management problem in `MergeTreeReadPool`. #8791 (Vladimir Chebotarev)
- Fix `toDecimal*OrNull()` functions family when called with string `e`. Fixes #8312 #8764 (Artem Zuikov)
- Make sure that `FORMAT Null` sends no data to the client. #8767 (Alexander Kuzmenkov)
- Fix bug that timestamp in `LiveViewBlockInputStream` will not updated. `LIVE VIEW` is an experimental feature. #8644 (vxider) #8625 (vxider)
- Fixed `ALTER MODIFY TTL` wrong behavior which did not allow to delete old TTL expressions. #8422 (Vladimir Chebotarev)
- Fixed UBSan report in `MergeTreeIndexSet`. This fixes #9250 #9365 (alexey-milovidov)
- Fixed the behaviour of `match` and `extract` functions when haystack has zero bytes. The behaviour was wrong when haystack was constant. This fixes #9160 #9163 (alexey-milovidov) #9345 (alexey-milovidov)
- Avoid throwing from destructor in Apache Avro 3rd-party library. #9066 (Andrew Onyshchuk)
- Don't commit a batch polled from `Kafka` partially as it can lead to holes in data. #8876 (filimonov)
- Fix `joinGet` with nullable return types. <https://github.com/ClickHouse/ClickHouse/issues/8919> #9014 (Amos Bird)
- Fix data incompatibility when compressed with `T64` codec. #9016 (Artem Zuikov) Fix data type ids in `T64` compression codec that leads to wrong (de)compression in affected versions. #9033 (Artem Zuikov)
- Add setting `enable_early_constant_folding` and disable it in some cases that leads to errors. #9010 (Artem Zuikov)
- Fix pushdown predicate optimizer with `VIEW` and enable the test #9011 (Winter Zhang)
- Fix segfault in `Merge` tables, that can happen when reading from `File` storages #9387 (tavplubix)
- Added a check for storage policy in `ATTACH PARTITION FROM`, `REPLACE PARTITION`, `MOVE TO TABLE`. Otherwise it could make data of part inaccessible after restart and prevent ClickHouse to start. #9383 (Vladimir Chebotarev)
- Fix alters if there is TTL set for table. #8800 (Anton Popov)
- Fix race condition that can happen when `SYSTEM RELOAD ALL DICTIONARIES` is executed while some dictionary is being modified/added/removed. #8801 (Vitaly Baranov)
- In previous versions `Memory` database engine use empty data path, so tables are created in `path` directory (e.g. `/var/lib/clickhouse/`), not in data directory of database (e.g. `/var/lib/clickhouse/db_name`). #8753 (tavplubix)
- Fixed wrong log messages about missing default disk or policy. #9530 (Vladimir Chebotarev)
- Fix `not(has())` for the `bloom_filter` index of array types. #9407 (achimbab)
- Allow first column(s) in a table with `Log` engine be an alias #9231 (Ivan)
- Fix order of ranges while reading from `MergeTree` table in one thread. It could lead to exceptions from `MergeTreeRangeReader` or wrong query results. #9050 (Anton Popov)
- Make `reinterpretAsFixedString` to return `FixedSize` instead of `String`. #9052 (Andrew Onyshchuk)
- Avoid extremely rare cases when the user can get wrong error message (`Success` instead of detailed error description). #9457 (alexey-milovidov)
- Do not crash when using `Template` format with empty row template. #8785 (Alexander Kuzmenkov)
- Metadata files for system tables could be created in wrong place #8653 (tavplubix) Fixes #8581.
- Fix data race on `exception_ptr` in cache dictionary #8303. #9379 (Nikita Mikhaylov)
- Do not throw an exception for query `ATTACH TABLE IF NOT EXISTS`. Previously it was thrown if table already exists, despite the `IF NOT EXISTS` clause. #8967 (Anton Popov)
- Fixed missing closing paren in exception message. #8811 (alexey-milovidov)
- Avoid message `Possible deadlock avoided` at the startup of `clickhouse-client` in interactive mode. #9455 (alexey-milovidov)
- Fixed the issue when padding at the end of base64 encoded value can be malformed. Update base64 library. This fixes #9491, closes #9492 #9500 (alexey-milovidov)
- Prevent losing data in `Kafka` in rare cases when exception happens after reading suffix but before commit. Fixes #9378 #9507 (filimonov)
- Fixed exception in `DROP TABLE IF EXISTS` #8663 (Nikita Vasilev)
- Fix crash when a user tries to `ALTER MODIFY SETTING` for old-formatted `MergeTree` table engines family. #9435 (alesapin)

- Support for UInt64 numbers that don't fit in Int64 in JSON-related functions. Update SIMDJSON to master. This fixes #9209 #9344 (alexey-milovidov)
- Fixed execution of inverted predicates when non-strictly monotonic functional index is used. #9223 (Alexander Kazakov)
- Don't try to fold IN constant in GROUP BY #8868 (Amos Bird)
- Fix bug in ALTER DELETE mutations which leads to index corruption. This fixes #9019 and #8982. Additionally fix extremely rare race conditions in ReplicatedMergeTree ALTER queries. #9048 (alesapin)
- When the setting compile_expressions is enabled, you can get unexpected column in LLVMExecutableFunction when we use Nullable type #8910 (Guillaume Tassery)
- Multiple fixes for Kafka engine: 1) fix duplicates that were appearing during consumer group rebalance. 2) Fix rare 'holes' appeared when data were polled from several partitions with one poll and committed partially (now we always process / commit the whole polled block of messages). 3) Fix flushes by block size (before that only flushing by timeout was working properly). 4) better subscription procedure (with assignment feedback). 5) Make tests work faster (with default intervals and timeouts). Due to the fact that data was not flushed by block size before (as it should according to documentation), that PR may lead to some performance degradation with default settings (due to more often & tinier flushes which are less optimal). If you encounter the performance issue after that change - please increase kafka_max_block_size in the table to the bigger value (for example CREATE TABLE ...Engine=Kafka ... SETTINGS ... kafka_max_block_size=524288). Fixes #7259 #8917 (filimonov)
- Fix Parameter out of bound exception in some queries after PREWHERE optimizations. #8914 (Baudouin Giard)
- Fixed the case of mixed-constness of arguments of function arrayZip. #8705 (alexey-milovidov)
- When executing CREATE query, fold constant expressions in storage engine arguments. Replace empty database name with current database. Fixes #6508, #3492 #9262 (tavplubix)
- Now it's not possible to create or add columns with simple cyclic aliases like a DEFAULT b, b DEFAULT a. #9603 (alesapin)
- Fixed a bug with double move which may corrupt original part. This is relevant if you use ALTER TABLE MOVE #8680 (Vladimir Chebotarev)
- Allow interval identifier to correctly parse without backticks. Fixed issue when a query cannot be executed even if the interval identifier is enclosed in backticks or double quotes. This fixes #9124. #9142 (alexey-milovidov)
- Fixed fuzz test and incorrect behaviour of bitTestAll/bitTestAny functions. #9143 (alexey-milovidov)
- Fix possible crash/wrong number of rows in LIMIT n WITH TIES when there are a lot of rows equal to n'th row. #9464 (tavplubix)
- Fix mutations with parts written with enabled insert_quorum. #9463 (alesapin)
- Fix data race at destruction of Poco::HTTPServer. It could happen when server is started and immediately shut down. #9468 (Anton Popov)
- Fix bug in which a misleading error message was shown when running SHOW CREATE TABLE a_table_that_does_not_exist. #8899 (achulkov2)
- Fixed Parameters are out of bound exception in some rare cases when we have a constant in the SELECT clause when we have an ORDER BY and a LIMIT clause. #8892 (Guillaume Tassery)
- Fix mutations finalization, when already done mutation can have status is_done=0. #9217 (alesapin)
- Prevent from executing ALTER ADD INDEX for MergeTree tables with old syntax, because it doesn't work. #8822 (Mikhail Korotov)
- During server startup do not access table, which LIVE VIEW depends on, so server will be able to start. Also remove LIVE VIEW dependencies when detaching LIVE VIEW. LIVE VIEW is an experimental feature. #8824 (tavplubix)
- Fix possible segfault in MergeTreeRangeReader, while executing PREWHERE. #9106 (Anton Popov)
- Fix possible mismatched checksums with column TTLs. #9451 (Anton Popov)
- Fixed a bug when parts were not being moved in background by TTL rules in case when there is only one volume. #8672 (Vladimir Chebotarev)
- Fixed the issue Method createColumn() is not implemented for data type Set. This fixes #7799. #8674 (alexey-milovidov)
- Now we will try finalize mutations more frequently. #9427 (alesapin)
- Fix intDiv bv minus one constant #9351 (hc7)

... added by adding one constant `.. const ..`,

- Fix possible race condition in `BlockIO`. #9356 (Nikolai Kochetov)
- Fix bug leading to server termination when trying to use / drop `Kafka` table created with wrong parameters. #9513 (filimonov)
- Added workaround if OS returns wrong result for `timer_create` function. #8837 (alexey-milovidov)
- Fixed error in usage of `min_marks_for_seek` parameter. Fixed the error message when there is no sharding key in Distributed table and we try to skip unused shards. #8908 (Azat Khuzhin)

Improvement

- Implement `ALTER MODIFY/DROP` queries on top of mutations for `ReplicatedMergeTree*` engines family. Now `ALTERS` blocks only at the metadata update stage, and don't block after that. #8701 (alesapin)
- Add ability to rewrite `CROSS` to `INNER JOINs` with `WHERE` section containing unqualified names. #9512 (Artem Zuikov)
- Make `SHOW TABLES` and `SHOW DATABASES` queries support the `WHERE` expressions and `FROM/IN` #9076 (sundyli)
- Added a setting `deduplicate_blocks_in_dependent_materialized_views`. #9070 (urykhy)
- After recent changes MySQL client started to print binary strings in hex thereby making them not readable (#9032). The workaround in ClickHouse is to mark string columns as `UTF-8`, which is not always, but usually the case. #9079 (Yuriy Baranov)
- Add support of `String` and `FixedString` keys for `sumMap` #8903 (Baudouin Giard)
- Support string keys in `SummingMergeTree` maps #8933 (Baudouin Giard)
- Signal termination of thread to the thread pool even if the thread has thrown exception #8736 (Ding Xiang Fei)
- Allow to set `query_id` in `clickhouse-benchmark` #9416 (Anton Popov)
- Don't allow strange expressions in `ALTER TABLE ... PARTITION` partition query. This addresses #7192 #8835 (alexey-milovidov)
- The table `system.table_engines` now provides information about feature support (like `supports_ttl` or `supports_sort_order`). #8830 (Max Akhmedov)
- Enable `system.metric_log` by default. It will contain rows with values of `ProfileEvents`, `CurrentMetrics` collected with "collect_interval_milliseconds" interval (one second by default). The table is very small (usually in order of megabytes) and collecting this data by default is reasonable. #9225 (alexey-milovidov)
- Initialize query profiler for all threads in a group, e.g. it allows to fully profile insert-queries. Fixes #6964 #8874 (Ivan)
- Now temporary `LIVE VIEW` is created by `CREATE LIVE VIEW name WITH TIMEOUT [42] ...` instead of `CREATE TEMPORARY LIVE VIEW ...`, because the previous syntax was not consistent with `CREATE TEMPORARY TABLE ...` #9131 (tavplubix)
- Add `text_log.level` configuration parameter to limit entries that goes to `system.text_log` table #8809 (Azat Khuzhin)
- Allow to put downloaded part to a disks/volumes according to TTL rules #8598 (Vladimir Chebotarev)
- For external MySQL dictionaries, allow to mutualize MySQL connection pool to "share" them among dictionaries. This option significantly reduces the number of connections to MySQL servers. #9409 (Clément Rodriguez)
- Show nearest query execution time for quantiles in `clickhouse-benchmark` output instead of interpolated values. It's better to show values that correspond to the execution time of some queries. #8712 (alexey-milovidov)
- Possibility to add key & timestamp for the message when inserting data to Kafka. Fixes #7198 #8969 (filimonov)
- If server is run from terminal, highlight thread number, query id and log priority by colors. This is for improved readability of correlated log messages for developers. #8961 (alexey-milovidov)
- Better exception message while loading tables for `Ordinary` database. #9527 (alexey-milovidov)
- Implement `arraySlice` for arrays with aggregate function states. This fixes #9388 #9391 (alexey-milovidov)
- Allow constant functions and constant arrays to be used on the right side of IN operator. #8813 (Anton Popov)

- If zookeeper exception has happened while fetching data for system.replicas, display it in a separate column. This implements #9137 #9138 (alexey-milovidov)
- Atomically remove MergeTree data parts on destroy. #8402 (Vladimir Chebotarev)
- Support row-level security for Distributed tables. #8926 (Ivan)
- Now we recognize suffix (like KB, KiB...) in settings values. #8072 (Mikhail Korotov)
- Prevent out of memory while constructing result of a large JOIN. #8637 (Artem Zuikov)
- Added names of clusters to suggestions in interactive mode in clickhouse-client. #8709 (alexey-milovidov)
- Initialize query profiler for all threads in a group, e.g. it allows to fully profile insert-queries #8820 (Ivan)
- Added column exception_code in system.query_log table. #8770 (Mikhail Korotov)
- Enabled MySQL compatibility server on port 9004 in the default server configuration file. Fixed password generation command in the example in configuration. #8771 (Yuriy Baranov)
- Prevent abort on shutdown if the filesystem is readonly. This fixes #9094 #9100 (alexey-milovidov)
- Better exception message when length is required in HTTP POST query. #9453 (alexey-milovidov)
- Add _path and _file virtual columns to HDFS and File engines and hdfs and file table functions #8489 (Olga Khvostikova)
- Fix error Cannot find column while inserting into MATERIALIZED VIEW in case if new column was added to view's internal table. #8766 #8788 (vzakaznikov) #8788 #8806 (Nikolai Kochetov) #8803 (Nikolai Kochetov)
- Fix progress over native client-server protocol, by send progress after final update (like logs). This may be relevant only to some third-party tools that are using native protocol. #9495 (Azat Khuzhin)
- Add a system metric tracking the number of client connections using MySQL protocol (#9013). #9015 (Eugene Klimov)
- From now on, HTTP responses will have X-ClickHouse-Timezone header set to the same timezone value that SELECT timezone() would report. #9493 (Denis Glazachev)

Performance Improvement

- Improve performance of analysing index with IN #9261 (Anton Popov)
- Simpler and more efficient code in Logical Functions + code cleanups. A followup to #8718 #8728 (Alexander Kazakov)
- Overall performance improvement (in range of 5%..200% for affected queries) by ensuring even more strict aliasing with C++20 features. #9304 (Amos Bird)
- More strict aliasing for inner loops of comparison functions. #9327 (alexey-milovidov)
- More strict aliasing for inner loops of arithmetic functions. #9325 (alexey-milovidov)
- A ~3 times faster implementation for ColumnVector::replicate(), via which ColumnConst::convertToFullColumn() is implemented. Also will be useful in tests when materializing constants. #9293 (Alexander Kazakov)
- Another minor performance improvement to ColumnVector::replicate() (this speeds up the materialize function and higher order functions) an even further improvement to #9293 #9442 (Alexander Kazakov)
- Improved performance of stochasticLinearRegression aggregate function. This patch is contributed by Intel. #8652 (alexey-milovidov)
- Improve performance of reinterpretAsFixedString function. #9342 (alexey-milovidov)
- Do not send blocks to client for Null format in processors pipeline. #8797 (Nikolai Kochetov) #8767 (Alexander Kuzmenkov)

Build/Testing/Packaging Improvement

- Exception handling now works correctly on Windows Subsystem for Linux. See <https://github.com/ClickHouse-Extras/libunwind/pull/3> This fixes #6480 #9564 (sobolevsv)
- Replace readline with replxx for interactive line editing in clickhouse-client #8416 (Ivan)
- Better build time and less template instantiations in FunctionsComparison. #9324 (alexey-milovidov)
- Added integration with clang-tidy in CI. See also #6044 #9566 (alexey-milovidov)
- Now we link ClickHouse in CI using lld even for gcc. #9049 (alesapin)
- Allow to randomize thread scheduling and insert glitches when THREAD_FUZZER_* environment variables are set. This helps testing. #9459 (alexey-milovidov)

- Enable secure sockets in stateless tests #9288 (tavplubix)
- Make SPLIT_SHARED_LIBRARIES=OFF more robust #9156 (Azat Khuzhin)
- Make "performance_introspection_and_logging" test reliable to random server stuck. This may happen in CI environment. See also #9515 #9528 (alexey-milovidov)
- Validate XML in style check. #9550 (alexey-milovidov)
- Fixed race condition in test 00738_lock_for_inner_table. This test relied on sleep. #9555 (alexey-milovidov)
- Remove performance tests of type once. This is needed to run all performance tests in statistical comparison mode (more reliable). #9557 (alexey-milovidov)
- Added performance test for arithmetic functions. #9326 (alexey-milovidov)
- Added performance test for sumMap and sumMapWithOverflow aggregate functions. Follow-up for #8933 #8947 (alexey-milovidov)
- Ensure style of ErrorCodes by style check. #9370 (alexey-milovidov)
- Add script for tests history. #8796 (alesapin)
- Add GCC warning -Wsuggest-override to locate and fix all places where override keyword must be used. #8760 (kreuzerkrieg)
- Ignore weak symbol under Mac OS X because it must be defined #9538 (Deleted user)
- Normalize running time of some queries in performance tests. This is done in preparation to run all the performance tests in comparison mode. #9565 (alexey-milovidov)
- Fix some tests to support pytest with query tests #9062 (Ivan)
- Enable SSL in build with MSan, so server will not fail at startup when running stateless tests #9531 (tavplubix)
- Fix database substitution in test results #9384 (Ilya Yatsishin)
- Build fixes for miscellaneous platforms #9381 (proller) #8755 (proller) #8631 (proller)
- Added disks section to stateless-with-coverage test docker image #9213 (Pavel Kovalenko)
- Get rid of in-source-tree files when building with GRPC #9588 (Amos Bird)
- Slightly faster build time by removing SessionCleaner from Context. Make the code of SessionCleaner more simple. #9232 (alexey-milovidov)
- Updated checking for hung queries in clickhouse-test script #8858 (Alexander Kazakov)
- Removed some useless files from repository. #8843 (alexey-milovidov)
- Changed type of math perftests from once to loop. #8783 (Nikolai Kochetov)
- Add docker image which allows to build interactive code browser HTML report for our codebase. #8781 (alesapin) See [Woboq Code Browser](#)
- Suppress some test failures under MSan. #8780 (Alexander Kuzmenkov)
- Speedup "exception while insert" test. This test often time out in debug-with-coverage build. #8711 (alexey-milovidov)
- Updated libcxx and libcxxabi to master. In preparation to #9304 #9308 (alexey-milovidov)
- Fix flaky test 00910_zookeeper_test_alter_compression_codecs. #9525 (alexey-milovidov)
- Clean up duplicated linker flags. Make sure the linker won't look up an unexpected symbol. #9433 (Amos Bird)
- Add clickhouse-odbc driver into test images. This allows to test interaction of ClickHouse with ClickHouse via its own ODBC driver. #9348 (filimonov)
- Fix several bugs in unit tests. #9047 (alesapin)
- Enable -Wmissing/include-dirs GCC warning to eliminate all non-existing includes - mostly as a result of CMake scripting errors #8704 (kreuzerkrieg)
- Describe reasons if query profiler cannot work. This is intended for #9049 #9144 (alexey-milovidov)
- Update OpenSSL to upstream master. Fixed the issue when TLS connections may fail with the message OpenSSL SSL_read: error:14094438:SSL routines:ssl3_read_bytes:tlsv1 alert internal error and SSL Exception: error:2400006E:random number generator::error retrieving entropy. The issue was present in version 20.1. #8956 (alexey-milovidov)
- Update Dockerfile for server #8893 (Ilya Mazaev)
- Minor fixes in build-gcc-from-sources script #8774 (Michael Nacharov)
- Replace numbers to zeros in perftests where number column is not used. This will lead to more clean test results. #9600 (Nikolai Kochetov)
- Fix stack overflow issue when using initializer_list in Column constructors. #9367 (Deleted user)
- Upgrade librdkafka to v1.3.0. Enable bundled rdkafka and osasl libraries on Mac OS X. #9000 (Andrew)

- build fix on GCC 9.2.0 #9306 (vxider)

ClickHouse release v20.1

ClickHouse release v20.1.10.70, 2020-04-17

Bug Fix

- Fix rare possible exception Cannot drain connections: cancel first. #10239 (Nikolai Kochetov).
- Fixed bug where ClickHouse would throw 'Unknown function lambda.' error message when user tries to run ALTER UPDATE/DELETE on tables with ENGINE = Replicated*. Check for nondeterministic functions now handles lambda expressions correctly. #10237 (Alexander Kazakov).
- Fix parseDateTimeBestEffort for strings in RFC-2822 when day of week is Tuesday or Thursday. This fixes #10082. #10214 (alexey-milovidov).
- Fix column names of constants inside JOIN that may clash with names of constants outside of JOIN. #10207 (alexey-milovidov).
- Fix possible infinite query execution when the query actually should stop on LIMIT, while reading from infinite source like system.numbers or system.zeros. #10206 (Nikolai Kochetov).
- Fix move-to-prewhere optimization in presence of arrayJoin functions (in certain cases). This fixes #10092. #10195 (alexey-milovidov).
- Add the ability to relax the restriction on non-deterministic functions usage in mutations with allow_nondeterministic_mutations setting. #10186 (filimonov).
- Convert blocks if structure does not match on INSERT into table with Distributed engine. #10135 (Azat Khuzhin).
- Fix SIGSEGV on INSERT into Distributed table when its structure differs from the underlying tables. #10105 (Azat Khuzhin).
- Fix possible rows loss for queries with JOIN and UNION ALL. Fixes #9826, #10113. #10099 (Nikolai Kochetov).
- Add arguments check and support identifier arguments for MySQL Database Engine. #10077 (Winter Zhang).
- Fix bug in clickhouse dictionary source from localhost clickhouse server. The bug may lead to memory corruption if types in dictionary and source are not compatible. #10071 (alesapin).
- Fix error Cannot clone block with columns because block has 0 columns ... While executing GroupingAggregatedTransform. It happened when setting distributed_aggregation_memory_efficient was enabled, and distributed query read aggregating data with different level from different shards (mixed single and two level aggregation). #10063 (Nikolai Kochetov).
- Fix a segmentation fault that could occur in GROUP BY over string keys containing trailing zero bytes (#8636, #8925). #10025 (Alexander Kuzmenkov).
- Fix bug in which the necessary tables weren't retrieved at one of the processing stages of queries to some databases. Fixes #9699. #9949 (achulkov2).
- Fix 'Not found column in block' error when JOIN appears with TOTALS. Fixes #9839. #9939 (Artem Zuikov).
- Fix a bug with ON CLUSTER DDL queries freezing on server startup. #9927 (Gagan Arneja).
- Fix TRUNCATE for Join table engine (#9917). #9920 (Amos Bird).
- Fix 'scalar doesn't exist' error in ALTER queries (#9878). #9904 (Amos Bird).
- Fix race condition between drop and optimize in ReplicatedMergeTree. #9901 (alesapin).
- Fixed DeleteOnDestroy logic in ATTACH PART which could lead to automatic removal of attached part and added few tests. #9410 (Vladimir Chebotarev).

Build/Testing/Packaging Improvement

- Fix unit test collapsing_sorted_stream. #9367 (Deleted user).

ClickHouse release v20.1.9.54, 2020-03-28

Bug Fix

- Fix 'Different expressions with the same alias' error when query has `PREWHERE` and `WHERE` on distributed table and `SET distributed_product_mode = 'local'`. #9871 (Artem Zuikov).
- Fix mutations excessive memory consumption for tables with a composite primary key. This fixes #9850. #9860 (alesapin).
- For `INSERT` queries shard now clamps the settings got from the initiator to the shard's constraints instead of throwing an exception. This fix allows to send `INSERT` queries to a shard with another constraints. This change improves fix #9447. #9852 (Vitaly Baranov).
- Fix possible exception Got 0 in totals chunk, expected 1 on client. It happened for queries with `JOIN` in case if right joined table had zero rows. Example: `select * from system.one t1 join system.one t2 on t1.dummy = t2.dummy limit 0 FORMAT TabSeparated;`. Fixes #9777. #9823 (Nikolai Kochetov).
- Fix `SIGSEGV` with `optimize_skip_unused_shards` when type cannot be converted. #9804 (Azat Khuzhin).
- Fixed a few cases when timezone of the function argument wasn't used properly. #9574 (Vasily Nemkov).

Improvement

- Remove `ORDER BY` stage from mutations because we read from a single ordered part in a single thread. Also add check that the order of rows in mutation is ordered in sorting key order and this order is not violated. #9886 (alesapin).

Build/Testing/Packaging Improvement

- Clean up duplicated linker flags. Make sure the linker won't look up an unexpected symbol. #9433 (Amos Bird).

ClickHouse release v20.1.8.41, 2020-03-20

Bug Fix

- Fix possible permanent `Cannot schedule a task` error (due to unhandled exception in `ParallelAggregatingBlockInputStream::Handler::onFinish/onFinishThread`). This fixes #6833. #9154 (Azat Khuzhin)
- Fix excessive memory consumption in `ALTER` queries (mutations). This fixes #9533 and #9670. #9754 (alesapin)
- Fix bug in backquoting in external dictionaries DDL. This fixes #9619. #9734 (alesapin)

ClickHouse release v20.1.7.38, 2020-03-18

Bug Fix

- Fixed incorrect internal function names for `sumKahan` and `sumWithOverflow`. It lead to exception while using this functions in remote queries. #9636 (Azat Khuzhin). This issue was in all ClickHouse releases.
- Allow `ALTER ON CLUSTER` of `Distributed` tables with internal replication. This fixes #3268. #9617 (shinoi2). This issue was in all ClickHouse releases.
- Fix possible exceptions `Size of filter doesn't match size of column` and `Invalid number of rows in Chunk` in `MergeTreeRangeReader`. They could appear while executing `PREWHERE` in some cases. Fixes #9132. #9612 (Anton Popov)
- Fixed the issue: `timezone` was not preserved if you write a simple arithmetic expression like `time + 1` (in contrast to an expression like `time + INTERVAL 1 SECOND`). This fixes #5743. #9323 (alexey-milovidov). This issue was in all ClickHouse releases.
- Now it's not possible to create or add columns with simple cyclic aliases like `a DEFAULT b, b DEFAULT a`. #9603 (alesapin)
- Fixed the issue when padding at the end of base64 encoded value can be malformed. Update base64 library. This fixes #9491, closes #9492 #9500 (alexey-milovidov)
- Fix data race at destruction of `Poco::HTTPServer`. It could happen when server is started and immediately shut down. #9468 (Anton Popov)
- Fix possible crash/wrong number of rows in `LIMIT n WITH TIES` when there are a lot of rows equal to `n'th` row. #9464 (tavplubix)
- Fix possible mismatched checksums with column TTLs. #9451 (Anton Popov)

- Fix crash when a user tries to `ALTER MODIFY SETTING` for old-formatted MergeTree table engines family.
[#9435 \(alesapin\)](#)
- Now we will try finalize mutations more frequently. [#9427 \(alesapin\)](#)
- Fix replication protocol incompatibility introduced in [#8598](#). [#9412 \(alesapin\)](#)
- Fix `not(has())` for the bloom_filter index of array types. [#9407 \(achimbab\)](#)
- Fixed the behaviour of `match` and `extract` functions when haystack has zero bytes. The behaviour was wrong when haystack was constant. This fixes [#9160](#) [#9163 \(alexey-milovidov\)](#) [#9345 \(alexey-milovidov\)](#)

Build/Testing/Packaging Improvement

- Exception handling now works correctly on Windows Subsystem for Linux. See <https://github.com/ClickHouse-Extras/libunwind/pull/3> This fixes [#6480](#) [#9564 \(sobolevsv\)](#)

ClickHouse release v20.1.6.30, 2020-03-05

Bug Fix

- Fix data incompatibility when compressed with `T64` codec.
[#9039 \(abyss7\)](#)
- Fix order of ranges while reading from MergeTree table in one thread. Fixes [#8964](#).
[#9050 \(Curtiz\)](#)
- Fix possible segfault in `MergeTreeRangeReader`, while executing `PREWHERE`. Fixes [#9064](#).
[#9106 \(Curtiz\)](#)
- Fix `reinterpretAsFixedString` to return `FixedString` instead of `String`.
[#9052 \(oandrew\)](#)
- Fix `joinGet` with nullable return types. Fixes [#8919](#)
[#9014 \(amosbird\)](#)
- Fix fuzz test and incorrect behaviour of `bitTestAll`/`bitTestAny` functions.
[#9143 \(alexey-milovidov\)](#)
- Fix the behaviour of `match` and `extract` functions when haystack has zero bytes. The behaviour was wrong when haystack was constant. Fixes [#9160](#)
[#9163 \(alexey-milovidov\)](#)
- Fixed execution of inverted predicates when non-strictly monotonic functional index is used. Fixes [#9034](#)
[#9223 \(Akazz\)](#)
- Allow to rewrite `CROSS` to `INNER JOIN` if there's `[NOT] LIKE` operator in `WHERE` section. Fixes [#9191](#)
[#9229 \(4ertus2\)](#)
- Allow first column(s) in a table with Log engine be an alias.
[#9231 \(abyss7\)](#)
- Allow comma join with `IN()` inside. Fixes [#7314](#).
[#9251 \(4ertus2\)](#)
- Improve `ALTER MODIFY/ADD` queries logic. Now you cannot `ADD` column without type, `MODIFY` default expression doesn't change type of column and `MODIFY` type doesn't loose default expression value. Fixes [#8669](#).
[#9227 \(alesapin\)](#)
- Fix mutations finalization, when already done mutation can have status `is_done=0`.
[#9217 \(alesapin\)](#)
- Support "Processors" pipeline for `system.numbers` and `system.numbers_mt`. This also fixes the bug when `max_execution_time` is not respected.
[#7796 \(KochetovNicolai\)](#)
- Fix wrong counting of `DictCacheKeysRequestedFound` metric.
[#9411 \(nikitamikhaylov\)](#)
- Added a check for storage policy in `ATTACH PARTITION FROM`, `REPLACE PARTITION`, `MOVE TO TABLE` which otherwise could make data of part inaccessible after restart and prevent ClickHouse to start.
[#9383 \(excitoon\)](#)

- Fixed UBSan report in `MergeTreeIndexSet`. This fixes #9250
#9365 (alexey-milovidov)
- Fix possible datarace in `BlockIO`.
#9356 (KochetovNicolai)
- Support for `UInt64` numbers that don't fit in `Int64` in JSON-related functions. Update `SIMDJSON` to master. This fixes #9209
#9344 (alexey-milovidov)
- Fix the issue when the amount of free space is not calculated correctly if the data directory is mounted to a separate device. For default disk calculate the free space from data subdirectory. This fixes #7441
#9257 (millb)
- Fix the issue when TLS connections may fail with the message `OpenSSL SSL_read: error:14094438:SSL routines:ssl3_read_bytes:tlsv1 alert internal error` and `SSL Exception: error:2400006E:random number generator::error retrieving entropy`. Update OpenSSL to upstream master.
#8956 (alexey-milovidov)
- When executing `CREATE` query, fold constant expressions in storage engine arguments. Replace empty database name with current database. Fixes #6508, #3492. Also fix check for local address in `ClickHouseDictionarySource`.
#9262 (tabplubix)
- Fix segfault in `StorageMerge`, which can happen when reading from `StorageFile`.
#9387 (tabplubix)
- Prevent losing data in `Kafka` in rare cases when exception happens after reading suffix but before commit. Fixes #9378. Related: #7175
#9507 (filimonov)
- Fix bug leading to server termination when trying to use / drop `Kafka` table created with wrong parameters. Fixes #9494. Incorporates #9507.
#9513 (filimonov)

New Feature

- Add `deduplicate_blocks_in_dependent_materialized_views` option to control the behaviour of idempotent inserts into tables with materialized views. This new feature was added to the bugfix release by a special request from Altinity.
#9070 (urykhy)

ClickHouse release v20.1.2.4, 2020-01-22

Backward Incompatible Change

- Make the setting `merge_tree_uniform_read_distribution` obsolete. The server still recognizes this setting but it has no effect. #8308 (alexey-milovidov)
- Changed return type of the function `greatCircleDistance` to `Float32` because now the result of calculation is `Float32`. #7993 (alexey-milovidov)
- Now it's expected that query parameters are represented in "escaped" format. For example, to pass string `a<tab>b` you have to write `a\|b` or `a\<tab>b` and respectively, `a%5Ct%09b` or `a%5C%09b` in URL. This is needed to add the possibility to pass NULL as `\N`. This fixes #7488. #8517 (alexey-milovidov)
- Enable `use_minimalistic_part_header_in_zookeeper` setting for `ReplicatedMergeTree` by default. This will significantly reduce amount of data stored in ZooKeeper. This setting is supported since version 19.1 and we already use it in production in multiple services without any issues for more than half a year. Disable this setting if you have a chance to downgrade to versions older than 19.1. #6850 (alexey-milovidov)
- Data skipping indices are production ready and enabled by default. The settings `allow_experimental_data_skipping_indices`, `allow_experimental_cross_to_join_conversion` and `allow_experimental_multiple_joins_emulation` are now obsolete and do nothing. #7974 (alexey-milovidov)
- Add new `ANY JOIN` logic for `StorageJoin` consistent with `JOIN` operation. To upgrade without changes in behaviour you need add `SETTINGS any_join_distinct_right_table_keys = 1` to Engine Join tables metadata or recreate these tables after upgrade. #8400 (Artem Zuikov)
- Require server to be restarted to apply the changes in loading configuration. This is a temporary

Requires server to be restarted to apply the changes in logging configuration. This is a temporary workaround to avoid the bug where the server logs to a deleted log file (see #8696). #8707 (Alexander Kuzmenkov)

New Feature

- Added information about part paths to `system.merges`. #8043 (Vladimir Chebotarev)
- Add ability to execute `SYSTEM RELOAD DICTIONARY` query in `ON CLUSTER` mode. #8288 (Guillaume Tassery)
- Add ability to execute `CREATE DICTIONARY` queries in `ON CLUSTER` mode. #8163 (alesapin)
- Now user's profile in `users.xml` can inherit multiple profiles. #8343 (Mikhail f. Shiryaev)
- Added `system.stack_trace` table that allows to look at stack traces of all server threads. This is useful for developers to introspect server state. This fixes #7576. #8344 (alexey-milovidov)
- Add `DateTime64` datatype with configurable sub-second precision. #7170 (Vasily Nemkov)
- Add table function `clusterAllReplicas` which allows to query all the nodes in the cluster. #8493 (kiran sunkari)
- Add aggregate function `categoricalInformationValue` which calculates the information value of a discrete feature. #8117 (hcz)
- Speed up parsing of data files in `CSV`, `TSV` and `JSONEachRow` format by doing it in parallel. #7780 (Alexander Kuzmenkov)
- Add function `bankerRound` which performs banker's rounding. #8112 (hcz)
- Support more languages in embedded dictionary for region names: 'ru', 'en', 'ua', 'uk', 'by', 'kz', 'tr', 'de', 'uz', 'lv', 'lt', 'et', 'pt', 'he', 'vi'. #8189 (alexey-milovidov)
- Improvements in consistency of `ANY JOIN` logic. Now `t1 ANY LEFT JOIN t2` equals `t2 ANY RIGHT JOIN t1`. #7665 (Artem Zuikov)
- Add setting `any_join_distinct_right_table_keys` which enables old behaviour for `ANY INNER JOIN`. #7665 (Artem Zuikov)
- Add new `SEMI` and `ANTI JOIN`. Old `ANY INNER JOIN` behaviour now available as `SEMI LEFT JOIN`. #7665 (Artem Zuikov)
- Added `Distributed` format for `File` engine and `file` table function which allows to read from `.bin` files generated by asynchronous inserts into `Distributed` table. #8535 (Nikolai Kochetov)
- Add optional reset column argument for `runningAccumulate` which allows to reset aggregation results for each new key value. #8326 (Sergey Kononenko)
- Add ability to use ClickHouse as Prometheus endpoint. #7900 (vdimir)
- Add section `<remote_url_allow_hosts>` in `config.xml` which restricts allowed hosts for remote table engines and table functions `URL`, `S3`, `HDFS`. #7154 (Mikhail Korotov)
- Added function `greatCircleAngle` which calculates the distance on a sphere in degrees. #8105 (alexey-milovidov)
- Changed Earth radius to be consistent with H3 library. #8105 (alexey-milovidov)
- Added `JSONCompactEachRow` and `JSONCompactEachRowWithNamesAndTypes` formats for input and output. #7841 (Mikhail Korotov)
- Added feature for file-related table engines and table functions (`File`, `S3`, `URL`, `HDFS`) which allows to read and write `gzip` files based on additional engine parameter or file extension. #7840 (Andrey Bodrov)
- Added the `randomASCII(length)` function, generating a string with a random set of ASCII printable characters. #8401 (BayoNet)
- Added function `JSONExtractArrayRaw` which returns an array on unparsed json array elements from `JSON` string. #8081 (Oleg Matrokhin)
- Add `arrayZip` function which allows to combine multiple arrays of equal lengths into one array of tuples. #8149 (Winter Zhang)
- Add ability to move data between disks according to configured `TTL`-expressions for `*MergeTree` table engines family. #8140 (Vladimir Chebotarev)
- Added new aggregate function `avgWeighted` which allows to calculate weighted average. #7898 (Andrey Bodrov)
- Now parallel parsing is enabled by default for `TSV`, `TSKV`, `CSV` and `JSONEachRow` formats. #7894 (Nikita Mikhaylov)
- Add several geo functions from `H3` library: `h3GetResolution`, `h3EdgeAngle`, `h3EdgeLength`, `h3IsValid` and

h3kRing. #8034 (Konstantin Malanchev)

- Added support for brotli (br) compression in file-related storages and table functions. This fixes #8156. #8526 (alexey-milovidov)
- Add `groupBit*` functions for the `SimpleAggregationFunction` type. #8485 (Guillaume Tassery)

Bug Fix

- Fix rename of tables with `Distributed` engine. Fixes issue #7868. #8306 (tavplubix)
- Now dictionaries support `EXPRESSION` for attributes in arbitrary string in non-ClickHouse SQL dialect. #8098 (alesapin)
- Fix broken `INSERT SELECT FROM mysql(...)` query. This fixes #8070 and #7960. #8234 (tavplubix)
- Fix error "Mismatch column sizes" when inserting default Tuple from `JSONEachRow`. This fixes #5653. #8606 (tavplubix)
- Now an exception will be thrown in case of using `WITH TIES` alongside `LIMIT BY`. Also add ability to use `TOP` with `LIMIT BY`. This fixes #7472. #7637 (Nikita Mikhaylov)
- Fix unintended dependency from fresh glibc version in `clickhouse-odbc-bridge` binary. #8046 (Amos Bird)
- Fix bug in check function of `*MergeTree` engines family. Now it doesn't fail in case when we have equal amount of rows in last granule and last mark (non-final). #8047 (alesapin)
- Fix insert into `Enum*` columns after `ALTER` query, when underlying numeric type is equal to table specified type. This fixes #7836. #7908 (Anton Popov)
- Allowed non-constant negative "size" argument for function `substring`. It was not allowed by mistake. This fixes #4832. #7703 (alexey-milovidov)
- Fix parsing bug when wrong number of arguments passed to `(O)JDBC` table engine. #7709 (alesapin)
- Using command name of the running clickhouse process when sending logs to syslog. In previous versions, empty string was used instead of command name. #8460 (Michael Nacharov)
- Fix check of allowed hosts for `localhost`. This PR fixes the solution provided in #8241. #8342 (Vitaly Baranov)
- Fix rare crash in `argMin` and `argMax` functions for long string arguments, when result is used in `runningAccumulate` function. This fixes #8325 #8341 (dinosaur)
- Fix memory overcommit for tables with `Buffer` engine. #8345 (Azat Khuzhin)
- Fixed potential bug in functions that can take `NULL` as one of the arguments and return non-`NULL`. #8196 (alexey-milovidov)
- Better metrics calculations in thread pool for background processes for `MergeTree` table engines. #8194 (Vladimir Chebotarev)
- Fix function `IN` inside `WHERE` statement when row-level table filter is present. Fixes #6687 #8357 (Ivan)
- Now an exception is thrown if the integral value is not parsed completely for settings values. #7678 (Mikhail Korotov)
- Fix exception when aggregate function is used in query to distributed table with more than two local shards. #8164 (小路)
- Now bloom filter can handle zero length arrays and doesn't perform redundant calculations. #8242 (achimbab)
- Fixed checking if a client host is allowed by matching the client host to `host_regexp` specified in `users.xml`. #8241 (Vitaly Baranov)
- Relax ambiguous column check that leads to false positives in multiple `JOIN ON` section. #8385 (Artem Zuikov)
- Fixed possible server crash (`std::terminate`) when the server cannot send or write data in `JSON` or `XML` format with values of `String` data type (that require `UTF-8` validation) or when compressing result data with Brotli algorithm or in some other rare cases. This fixes #7603 #8384 (alexey-milovidov)
- Fix race condition in `StorageDistributedDirectoryMonitor` found by CI. This fixes #8364. #8383 (Nikolai Kochetov)
- Now background merges in `*MergeTree` table engines family preserve storage policy volume order more accurately. #8549 (Vladimir Chebotarev)
- Now table engine `Kafka` works properly with `Native` format. This fixes #6731 #7337 #8003. #8016 (filimonov)
- Fixed formats with headers (like `CSVWithNames`) which were throwing exception about EOF for table

- engine Kafka. #8016 (tilimonov)
- Fixed a bug with making set from subquery in right part of IN section. This fixes #5767 and #2542. #7755 (Nikita Mikhaylov)
- Fix possible crash while reading from storage File. #7756 (Nikolai Kochetov)
- Fixed reading of the files in Parquet format containing columns of type list. #8334 (maxulan)
- Fix error Not found column for distributed queries with PREWHERE condition dependent on sampling key if max_parallel_replicas > 1. #7913 (Nikolai Kochetov)
- Fix error Not found column if query used PREWHERE dependent on table's alias and the result set was empty because of primary key condition. #7911 (Nikolai Kochetov)
- Fixed return type for functions rand and randConstant in case of Nullable argument. Now functions always return UInt32 and never Nullable(UInt32). #8204 (Nikolai Kochetov)
- Disabled predicate push-down for WITH FILL expression. This fixes #7784. #7789 (Winter Zhang)
- Fixed incorrect count() result for SummingMergeTree when FINAL section is used. #3280 #7786 (Nikita Mikhaylov)
- Fix possible incorrect result for constant functions from remote servers. It happened for queries with functions like version(), uptime(), etc. which returns different constant values for different servers. This fixes #7666. #7689 (Nikolai Kochetov)
- Fix complicated bug in push-down predicate optimization which leads to wrong results. This fixes a lot of issues on push-down predicate optimization. #8503 (Winter Zhang)
- Fix crash in CREATE TABLE .. AS dictionary query. #8508 (Azat Khuzhin)
- Several improvements ClickHouse grammar in .g4 file. #8294 (taiyang-li)
- Fix bug that leads to crashes in JOINs with tables with engine Join. This fixes #7556 #8254 #7915 #8100. #8298 (Artem Zuikov)
- Fix redundant dictionaries reload on CREATE DATABASE. #7916 (Azat Khuzhin)
- Limit maximum number of streams for read from StorageFile and StorageHDFS. Fixes <https://github.com/ClickHouse/ClickHouse/issues/7650>. #7981 (alesapin)
- Fix bug in ALTER ... MODIFY ... CODEC query, when user specify both default expression and codec. Fixes 8593. #8614 (alesapin)
- Fix error in background merge of columns with SimpleAggregateFunction(LowCardinality) type. #8613 (Nikolai Kochetov)
- Fixed type check in function toDateTime64. #8375 (Vasily Nemkov)
- Now server do not crash on LEFT or FULL JOIN with and Join engine and unsupported join_use_nulls settings. #8479 (Artem Zuikov)
- Now DROP DICTIONARY IF EXISTS db.dict query doesn't throw exception if db doesn't exist. #8185 (Vitaly Baranov)
- Fix possible crashes in table functions (file, mysql, remote) caused by usage of reference to removed IStorage object. Fix incorrect parsing of columns specified at insertion into table function. #7762 (tavplubix)
- Ensure network be up before starting clickhouse-server. This fixes #7507. #8570 (Zhichang Yu)
- Fix timeouts handling for secure connections, so queries doesn't hang indefinitely. This fixes #8126. #8128 (alexey-milovidov)
- Fix clickhouse-copier's redundant contention between concurrent workers. #7816 (Ding Xiang Fei)
- Now mutations doesn't skip attached parts, even if their mutation version were larger than current mutation version. #7812 (Zhichang Yu) #8250 (alesapin)
- Ignore redundant copies of *MergeTree data parts after move to another disk and server restart. #7810 (Vladimir Chebotarev)
- Fix crash in FULL JOIN with LowCardinality in JOIN key. #8252 (Artem Zuikov)
- Forbidden to use column name more than once in insert query like INSERT INTO tbl (x, y, x). This fixes #5465, #7681. #7685 (alesapin)
- Added fallback for detection the number of physical CPU cores for unknown CPUs (using the number of logical CPU cores). This fixes #5239. #7726 (alexey-milovidov)
- Fix There's no column error for materialized and alias columns. #8210 (Artem Zuikov)
- Fixed sever crash when EXISTS query was used without TABLE or DICTIONARY qualifier. Just like EXISTS t. This fixes #8172. This bug was introduced in version 19.17. #8213 (alexey-milovidov)
- Fix rare bug with error "Sizes of columns doesn't match" that might appear when using

`SimpleAggregateFunction` column. #7790 (Boris Granveaud)

- Fix bug where user with empty `allow_databases` got access to all databases (and same for `allow_dictionaries`). #7793 (DeifyTheGod)
- Fix client crash when server already disconnected from client. #8071 (Azat Khuzhin)
- Fix `ORDER BY` behaviour in case of sorting by primary key prefix and non primary key suffix. #7759 (Anton Popov)
- Check if qualified column present in the table. This fixes #6836. #7758 (Artem Zuikov)
- Fixed behavior with `ALTER MOVE` ran immediately after merge finish moves superpart of specified. Fixes #8103. #8104 (Vladimir Chebotarev)
- Fix possible server crash while using `UNION` with different number of columns. Fixes #7279. #7929 (Nikolai Kochetov)
- Fix size of result substring for function `substr` with negative size. #8589 (Nikolai Kochetov)
- Now server does not execute part mutation in `MergeTree` if there are not enough free threads in background pool. #8588 (tavplubix)
- Fix a minor typo on formatting `UNION ALL AST`. #7999 (litao91)
- Fixed incorrect bloom filter results for negative numbers. This fixes #8317. #8566 (Winter Zhang)
- Fixed potential buffer overflow in decompress. Malicious user can pass fabricated compressed data that will cause read after buffer. This issue was found by Eldar Zaitov from Yandex information security team. #8404 (alexey-milovidov)
- Fix incorrect result because of integers overflow in `arrayIntersect`. #7777 (Nikolai Kochetov)
- Now `OPTIMIZE TABLE` query will not wait for offline replicas to perform the operation. #8314 (javi santana)
- Fixed `ALTER TTL` parser for `Replicated*MergeTree` tables. #8318 (Vladimir Chebotarev)
- Fix communication between server and client, so server read temporary tables info after query failure. #8084 (Azat Khuzhin)
- Fix `bitmapAnd` function error when intersecting an aggregated bitmap and a scalar bitmap. #8082 (Yue Huang)
- Refine the definition of `ZXid` according to the ZooKeeper Programmer's Guide which fixes bug in `clickhouse-cluster-copier`. #8088 (Ding Xiang Fei)
- `odbc` table function now respects `external_table_functions_use_nulls` setting. #7506 (Vasily Nemkov)
- Fixed bug that lead to a rare data race. #8143 (Alexander Kazakov)
- Now `SYSTEM RELOAD DICTIONARY` reloads a dictionary completely, ignoring `update_field`. This fixes #7440. #8037 (Vitaly Baranov)
- Add ability to check if dictionary exists in create query. #8032 (alesapin)
- Fix `Float*` parsing in `Values` format. This fixes #7817. #7870 (tavplubix)
- Fix crash when we cannot reserve space in some background operations of `*MergeTree` table engines family. #7873 (Vladimir Chebotarev)
- Fix crash of merge operation when table contains `SimpleAggregateFunction(LowCardinality)` column. This fixes #8515. #8522 (Azat Khuzhin)
- Restore support of all ICU locales and add the ability to apply collations for constant expressions. Also add language name to `system.collations` table. #8051 (alesapin)
- Fix bug when external dictionaries with zero minimal lifetime (`LIFETIME(MIN 0 MAX N)`, `LIFETIME(N)`) don't update in background. #7983 (alesapin)
- Fix crash when external dictionary with ClickHouse source has subquery in query. #8351 (Nikolai Kochetov)
- Fix incorrect parsing of file extension in table with engine `URL`. This fixes #8157. #8419 (Andrey Bodrov)
- Fix `CHECK TABLE` query for `*MergeTree` tables without key. Fixes #7543. #7979 (alesapin)
- Fixed conversion of `Float64` to MySQL type. #8079 (Yuriy Baranov)
- Now if table was not completely dropped because of server crash, server will try to restore and load it. #8176 (tavplubix)
- Fixed crash in table function `file` while inserting into file that doesn't exist. Now in this case file would be created and then insert would be processed. #8177 (Olga Khvostikova)
- Fix rare deadlock which can happen when `trace_log` is enabled. #7838 (filimonov)

- Add ability to work with different types besides `Date` in `RangeHashed` external dictionary created from DDL query. Fixes [#7899](#). [#8275 \(alesapin\)](#)
- Fixes crash when `now64()` is called with result of another function. [#8270 \(Vasily Nemkov\)](#)
- Fixed bug with detecting client IP for connections through mysql wire protocol. [#7743 \(Dmitry Muzyka\)](#)
- Fix empty array handling in `arraySplit` function. This fixes [#7708](#). [#7747 \(hc\)](#)
- Fixed the issue when `pid-file` of another running clickhouse-server may be deleted. [#8487 \(Weiqing Xu\)](#)
- Fix dictionary reload if it has `invalidate_query`, which stopped updates and some exception on previous update tries. [#8029 \(alesapin\)](#)
- Fixed error in function `arrayReduce` that may lead to "double free" and error in aggregate function combinator `Resample` that may lead to memory leak. Added aggregate function `aggThrow`. This function can be used for testing purposes. [#8446 \(alexey-milovidov\)](#)

Improvement

- Improved logging when working with `S3` table engine. [#8251 \(Grigory Pervakov\)](#)
- Printed help message when no arguments are passed when calling `clickhouse-local`. This fixes [#5335](#). [#8230 \(Andrey Nagorny\)](#)
- Add setting `mutations_sync` which allows to wait `ALTER UPDATE/DELETE` queries synchronously. [#8237 \(alesapin\)](#)
- Allow to set up relative `user_files_path` in `config.xml` (in the way similar to `format_schema_path`). [#7632 \(hc\)](#)
- Add exception for illegal types for conversion functions with `-OrZero` postfix. [#7880 \(Andrey Konyaev\)](#)
- Simplify format of the header of data sending to a shard in a distributed query. [#8044 \(Vitaly Baranov\)](#)
- Live View table engine refactoring. [#8519 \(vzakaznikov\)](#)
- Add additional checks for external dictionaries created from DDL-queries. [#8127 \(alesapin\)](#)
- Fix error `Column ... already exists` while using `FINAL` and `SAMPLE` together, e.g. `select count() from table final sample 1/2`. Fixes [#5186](#). [#7907 \(Nikolai Kochetov\)](#)
- Now table the first argument of `joinGet` function can be table identifier. [#7707 \(Amos Bird\)](#)
- Allow using `MaterializedView` with subqueries above `Kafka` tables. [#8197 \(filimonov\)](#)
- Now background moves between disks run it the seprate thread pool. [#7670 \(Vladimir Chebotarev\)](#)
- `SYSTEM RELOAD DICTIONARY` now executes synchronously. [#8240 \(Vitaly Baranov\)](#)
- Stack traces now display physical addresses (offsets in object file) instead of virtual memory addresses (where the object file was loaded). That allows the use of `addr2line` when binary is position independent and ASLR is active. This fixes [#8360](#). [#8387 \(alexey-milovidov\)](#)
- Support new syntax for row-level security filters: `<table name='table_name'>...</table>`. Fixes [#5779](#). [#8381 \(Ivan\)](#)
- Now `cityHash` function can work with `Decimal` and `UUID` types. Fixes [#5184](#). [#7693 \(Mikhail Korotov\)](#)
- Removed fixed index granularity (it was 1024) from system logs because it's obsolete after implementation of adaptive granularity. [#7698 \(alexey-milovidov\)](#)
- Enabled MySQL compatibility server when ClickHouse is compiled without SSL. [#7852 \(Yuriy Baranov\)](#)
- Now server checksums distributed batches, which gives more verbose errors in case of corrupted data in batch. [#7914 \(Azat Khuzhin\)](#)
- Support `DROP DATABASE`, `DETACH TABLE`, `DROP TABLE` and `ATTACH TABLE` for MySQL database engine. [#8202 \(Winter Zhang\)](#)
- Add authentication in S3 table function and table engine. [#7623 \(Vladimir Chebotarev\)](#)
- Added check for extra parts of `MergeTree` at different disks, in order to not allow to miss data parts at undefined disks. [#8118 \(Vladimir Chebotarev\)](#)
- Enable SSL support for Mac client and server. [#8297 \(Ivan\)](#)
- Now ClickHouse can work as MySQL federated server (see <https://dev.mysql.com/doc/refman/5.7/en/federated-create-server.html>). [#7717 \(Maxim Fedotov\)](#)
- `clickhouse-client` now only enable bracketed-paste when multiquery is on and multiline is off. This fixes (#7757)[<https://github.com/ClickHouse/ClickHouse/issues/7757>]. [#7761 \(Amos Bird\)](#)
- Support `Array(Decimal)` in `if` function. [#7721 \(Artem Zuikov\)](#)
- Support Decimals in `arrayDifference`, `arrayCumSum` and `arrayCumSumNegative` functions. [#7724 \(Artem Zuikov\)](#)
- Added lifetime column to `system.dictionaries` table. [#6820](#) [#7727 \(kekekekule\)](#)

- Added `mechanic` column to `system.dictionaries` table. #7720 (Vladimir Chebotarev)
- Improved check for existing parts on different disks for *MergeTree table engines. Addresses #7660. #8440 (Vladimir Chebotarev)
 - Integration with AWS SDK for S3 interactions which allows to use all S3 features out of the box. #8011 (Pavel Kovalenko)
 - Added support for subqueries in Live View tables. #7792 (vzakaznikov)
 - Check for using Date or DateTime column from TTL expressions was removed. #7920 (Vladimir Chebotarev)
 - Information about disk was added to system.detached_parts table. #7833 (Vladimir Chebotarev)
 - Now settings `max_(table|partition)_size_to_drop` can be changed without a restart. #7779 (Grigory Pervakov)
 - Slightly better usability of error messages. Ask user not to remove the lines below Stack trace: #7897 (alexey-milovidov)
 - Better reading messages from Kafka engine in various formats after #7935. #8035 (Ivan)
 - Better compatibility with MySQL clients which don't support sha2_password auth plugin. #8036 (Yuriy Baranov)
 - Support more column types in MySQL compatibility server. #7975 (Yuriy Baranov)
 - Implement ORDER BY optimization for Merge, Buffer and Materialized View storages with underlying MergeTree tables. #8130 (Anton Popov)
 - Now we always use POSIX implementation of `getrandom` to have better compatibility with old kernels (< 3.17). #7940 (Amos Bird)
 - Better check for valid destination in a move TTL rule. #8410 (Vladimir Chebotarev)
 - Better checks for broken insert batches for Distributed table engine. #7933 (Azat Khuzhin)
 - Add column with array of parts name which mutations must process in future to system.mutations table. #8179 (alesapin)
 - Parallel merge sort optimization for processors. #8552 (Nikolai Kochetov)
 - The settings `mark_cache_min_lifetime` is now obsolete and does nothing. In previous versions, mark cache can grow in memory larger than `mark_cache_size` to accommodate data within `mark_cache_min_lifetime` seconds. That was leading to confusion and higher memory usage than expected, that is especially bad on memory constrained systems. If you will see performance degradation after installing this release, you should increase the `mark_cache_size`. #8484 (alexey-milovidov)
 - Preparation to use `tid` everywhere. This is needed for #7477. #8276 (alexey-milovidov)

Performance Improvement

- Performance optimizations in processors pipeline. #7988 (Nikolai Kochetov)
- Non-blocking updates of expired keys in cache dictionaries (with permission to read old ones). #8303 (Nikita Mikhaylov)
- Compile ClickHouse without `-fno-omit-frame-pointer` globally to spare one more register. #8097 (Amos Bird)
- Speedup `greatCircleDistance` function and add performance tests for it. #7307 (Olga Khvostikova)
- Improved performance of function `roundDown`. #8465 (alexey-milovidov)
- Improved performance of `max`, `min`, `argMin`, `argMax` for `DateTime64` data type. #8199 (Vasily Nemkov)
- Improved performance of sorting without a limit or with big limit and external sorting. #8545 (alexey-milovidov)
- Improved performance of formatting floating point numbers up to 6 times. #8542 (alexey-milovidov)
- Improved performance of `modulo` function. #7750 (Amos Bird)
- Optimized ORDER BY and merging with single column key. #8335 (alexey-milovidov)
- Better implementation for `arrayReduce`, `-Array` and `-State` combinators. #7710 (Amos Bird)
- Now `PREWHERE` should be optimized to be at least as efficient as `WHERE`. #7769 (Amos Bird)
- Improve the way `round` and `roundBankers` handling negative numbers. #8229 (hc2)
- Improved decoding performance of `DoubleDelta` and `Gorilla` codecs by roughly 30-40%. This fixes #7082. #8019 (Vasily Nemkov)
- Improved performance of `base64` related functions. #8444 (alexey-milovidov)
- Added a function `geoDistance`. It is similar to `greatCircleDistance` but uses approximation to WGS-84 ellipsoid model. The performance of both functions are near the same. #8086 (alexey-milovidov)

- Faster `min` and `max` aggregation functions for `Decimal` data type. #8144 (Artem Zuikov)
- Vectorize processing `arrayReduce`. #7608 (Amos Bird)
- `if` chains are now optimized as `multilf`. #8355 (kamalov-ruslan)
- Fix performance regression of Kafka table engine introduced in 19.15. This fixes #7261. #7935 (filimonov)
- Removed "pie" code generation that `gcc` from Debian packages occasionally brings by default. #8483 (alexey-milovidov)
- Parallel parsing data formats #6553 (Nikita Mikhaylov)
- Enable optimized parser of `Values` with expressions by default (`input_format_values_deduce_templates_of_expressions=1`). #8231 (tavplubix)

Build/Testing/Packaging Improvement

- Build fixes for ARM and in minimal mode. #8304 (proller)
- Add coverage file flush for `clickhouse-server` when `std::atexit` is not called. Also slightly improved logging in stateless tests with coverage. #8267 (alesapin)
- Update LLVM library in contrib. Avoid using LLVM from OS packages. #8258 (alexey-milovidov)
- Make bundled curl build fully quiet. #8232 #8203 (Pavel Kovalenko)
- Fix some `MemorySanitizer` warnings. #8235 (Alexander Kuzmenkov)
- Use `add_warning` and `no_warning` macros in `CMakeLists.txt`. #8604 (Ivan)
- Add support of Minio S3 Compatible object (<https://min.io/>) for better integration tests. #7863 #7875 (Pavel Kovalenko)
- Imported `libc` headers to contrib. It allows to make builds more consistent across various systems (only for `x86_64-linux-gnu`). #5773 (alexey-milovidov)
- Remove `-fPIC` from some libraries. #8464 (alexey-milovidov)
- Clean `CMakeLists.txt` for curl. See <https://github.com/ClickHouse/ClickHouse/pull/8011#issuecomment-569478910> #8459 (alexey-milovidov)
- Silent warnings in `CapNProto` library. #8220 (alexey-milovidov)
- Add performance tests for short string optimized hash tables. #7679 (Amos Bird)
- Now ClickHouse will build on AArch64 even if `MADV_FREE` is not available. This fixes #8027. #8243 (Amos Bird)
- Update `zlib-ng` to fix memory sanitizer problems. #7182 #8206 (Alexander Kuzmenkov)
- Enable internal MySQL library on non-Linux system, because usage of OS packages is very fragile and usually doesn't work at all. This fixes #5765. #8426 (alexey-milovidov)
- Fixed build on some systems after enabling `libc++`. This supersedes #8374. #8380 (alexey-milovidov)
- Make `Field` methods more type-safe to find more errors. #7386 #8209 (Alexander Kuzmenkov)
- Added missing files to the `libc-headers` submodule. #8507 (alexey-milovidov)
- Fix wrong `JSON` quoting in performance test output. #8497 (Nikolai Kochetov)
- Now stack trace is displayed for `std::exception` and `Poco::Exception`. In previous versions it was available only for `DB::Exception`. This improves diagnostics. #8501 (alexey-milovidov)
- Porting `clock_gettime` and `clock_nanosleep` for fresh glibc versions. #8054 (Amos Bird)
- Enable `part_log` in example config for developers. #8609 (alexey-milovidov)
- Fix async nature of reload in `01036_no_superfluous_dict_reload_on_create_database*`. #8111 (Azat Khuzhin)
- Fixed codec performance tests. #8615 (Vasily Nemkov)
- Add install scripts for `.tgz` build and documentation for them. #8612 #8591 (alesapin)
- Removed old `ZSTD` test (it was created in year 2016 to reproduce the bug that pre 1.0 version of ZSTD has had). This fixes #8618. #8619 (alexey-milovidov)
- Fixed build on Mac OS Catalina. #8600 (meo)
- Increased number of rows in codec performance tests to make results noticeable. #8574 (Vasily Nemkov)
- In debug builds, treat `LOGICAL_ERROR` exceptions as assertion failures, so that they are easier to notice. #8475 (Alexander Kuzmenkov)
- Make formats-related performance test more deterministic. #8477 (alexey-milovidov)
- Update `Iz4` to fix a `MemorySanitizer` failure. #8181 (Alexander Kuzmenkov)
- Suppress a known `MemorySanitizer` false positive in exception handling. #8182 (Alexander Kuzmenkov)

- Update gcc and g++ to version 9 in build/docker/build.sh # / 66 (LightSky)
- Add performance test case to test that PREWHERE is worse than WHERE. #7768 (Amos Bird)
- Progress towards fixing one flaky test. #8621 (alexey-milovidov)
- Avoid MemorySanitizer report for data from libunwind. #8539 (alexey-milovidov)
- Updated libc++ to the latest version. #8324 (alexey-milovidov)
- Build ICU library from sources. This fixes #6460. #8219 (alexey-milovidov)
- Switched from libressl to openssl. ClickHouse should support TLS 1.3 and SNI after this change. This fixes #8171. #8218 (alexey-milovidov)
- Fixed UBSan report when using chacha20_poly1305 from SSL (happens on connect to <https://yandex.ru/>). #8214 (alexey-milovidov)
- Fix mode of default password file for .deb linux distros. #8075 (proller)
- Improved expression for getting clickhouse-server PID in clickhouse-test. #8063 (Alexander Kazakov)
- Updated contrib/googletest to v1.10.0. #8587 (Alexander Burmak)
- Fixed ThreadSanitizer report in base64 library. Also updated this library to the latest version, but it doesn't matter. This fixes #8397. #8403 (alexey-milovidov)
- Fix 00600_replace_running_query for processors. #8272 (Nikolai Kochetov)
- Remove support for tcmalloc to make CMakeLists.txt simpler. #8310 (alexey-milovidov)
- Release gcc builds now use libc++ instead of libstdc++. Recently libc++ was used only with clang. This will improve consistency of build configurations and portability. #8311 (alexey-milovidov)
- Enable ICU library for build with MemorySanitizer. #8222 (alexey-milovidov)
- Suppress warnings from CapNProto library. #8224 (alexey-milovidov)
- Removed special cases of code for tcmalloc, because it's no longer supported. #8225 (alexey-milovidov)
- In CI coverage task, kill the server gracefully to allow it to save the coverage report. This fixes incomplete coverage reports we've been seeing lately. #8142 (alesapin)
- Performance tests for all codecs against Float64 and UInt64 values. #8349 (Vasily Nemkov)
- termcap is very much deprecated and lead to various problems (f.g. missing "up" cap and echoing ^J instead of multi line) . Favor terminfo or bundled ncurses. #7737 (Amos Bird)
- Fix test_storage_s3 integration test. #7734 (Nikolai Kochetov)
- Support StorageFile(<format>, null) to insert block into given format file without actually write to disk. This is required for performance tests. #8455 (Amos Bird)
- Added argument --print-time to functional tests which prints execution time per test. #8001 (Nikolai Kochetov)
- Added asserts to KeyCondition while evaluating RPN. This will fix warning from gcc-9. #8279 (alexey-milovidov)
- Dump cmake options in CI builds. #8273 (Alexander Kuzmenkov)
- Don't generate debug info for some fat libraries. #8271 (alexey-milovidov)
- Make log_to_console.xml always log to stderr, regardless of is it interactive or not. #8395 (Alexander Kuzmenkov)
- Removed some unused features from clickhouse-performance-test tool. #8555 (alexey-milovidov)
- Now we will also search for lld-X with corresponding clang-X version. #8092 (alesapin)
- Parquet build improvement. #8421 (maxulan)
- More GCC warnings #8221 (kreuzerkrieg)
- Package for Arch Linux now allows to run ClickHouse server, and not only client. #8534 (Vladimir Chebotarev)
- Fix test with processors. Tiny performance fixes. #7672 (Nikolai Kochetov)
- Update contrib/protobuf. #8256 (Matvey V. Kornilov)
- In preparation of switching to c++20 as a new year celebration. "May the C++ force be with ClickHouse." #8447 (Amos Bird)

Experimental Feature

- Added experimental setting min_bytes_to_use_mmap_io. It allows to read big files without copying data from kernel to userspace. The setting is disabled by default. Recommended threshold is about 64 MB, because mmap/munmap is slow. #8520 (alexey-milovidov)
- Reworked quotas as a part of access control system. Added new table system.quotas, new functions currentQuota, currentQuotaKey, new SQL syntax CREATE QUOTA, ALTER QUOTA, DROP QUOTA, SHOW QUOTA.

#7257 (Vitaly Baranov)

- Allow skipping unknown settings with warnings instead of throwing exceptions. #7653 (Vitaly Baranov)
- Reworked row policies as a part of access control system. Added new table system.row_policies, new function currentRowPolicies(), new SQL syntax CREATE POLICY, ALTER POLICY, DROP POLICY, SHOW CREATE POLICY, SHOW POLICIES. #7808 (Vitaly Baranov)

Security Fix

- Fixed the possibility of reading directories structure in tables with File table engine. This fixes #8536. #8537 (alexey-milovidov)

Changelog for 2019

ClickHouse释放19.17

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 在解压缩固定潜在的缓冲区溢出。恶意用户可以传递制造的压缩数据，可能导致缓冲区后读取。这个问题是由Yandex信息安全部队的Eldar Zaitov发现的。 #8404 (阿列克谢-米洛维多夫)
- 修正了可能的服务器崩溃 (std::terminate) 当服务器不能发送或写入JSON或XML格式的数据与字符串数据类型的值（需要UTF-8验证），或者当压缩结果数据与Brotli算法或其他一些罕见的情况下。 #8384 (阿列克谢-米洛维多夫)
- 从clickhouse源固定字典 VIEW，现在阅读这样的字典不会导致错误 There is no query. #8351 (尼古拉*科切托夫)
- 修复了用户中指定的host _regexp是否允许客户端主机的检查。xml #8241, #8342 (维塔利*巴拉诺夫)
- RENAME TABLE 对于分布式表，现在在发送到分片之前重命名包含插入数据的文件夹。这解决了连续重命名的问题 tableA->tableB, tableC->tableA. #8306 (tavplubix)
- range_hashed DDL查询创建的外部字典现在允许任意数字类型的范围。 #8275 (阿利沙平)
- 固定 INSERT INTO table SELECT ... FROM mysql(...) 表功能。 #8234 (tavplubix)
- 修复段错误 INSERT INTO TABLE FUNCTION file() 同时插入到一个不存在的文件。现在在这种情况下，文件将被创建，然后插入将被处理。 #8177 (Olga Khvostikova)
- 修正了聚合位图和标量位图相交时的位图和错误。 #8082 (黄月)
- 修复段错误时 EXISTS 查询没有使用 TABLE 或 DICTIONARY 预选赛，就像 EXISTS t. #8213 (阿列克谢-米洛维多夫)
- 函数的固定返回类型 rand 和 randConstant 在可为空的参数的情况下。现在函数总是返回 UInt32 而且从来没有 Nullable(UInt32). #8204 (尼古拉*科切托夫)
- 固定 DROP DICTIONARY IF EXISTS db.dict，现在它不会抛出异常，如果 db 根本不存在 #8185 (维塔利*巴拉诺夫)
- 如果由于服务器崩溃而未完全删除表，服务器将尝试恢复并加载它 #8176 (tavplubix)
- 修正了一个简单的计数查询分布式表，如果有两个以上的分片本地表。 #8164 (小路)
- 修正了导致DB::BlockStreamProfileInfo::calculateRowsBeforeLimit数据竞赛的错误() #8143 (亚历山大*卡扎科夫)
- 固定 ALTER table MOVE part 在合并指定部件后立即执行，这可能导致移动指定部件合并到的部件。现在它正确地移动指定的部分。 #8104 (Vladimir Chebotarev)
- 字典的表达式现在可以指定为字符串。这对于从非ClickHouse源中提取数据时计算属性非常有用，因为它允许对这些表达式使用非ClickHouse语法。 #8098 (阿利沙平)
- 修正了一个非常罕见的比赛 clickhouse-copier 由于ZXid的溢出。 #8088 (丁香飞)
- 修复了查询失败后的错误（由于“Too many simultaneous queries”例如）它不会读取外部表信息，并且下一个请求会将此信息解释为下一个查询的开始，导致如下错误 Unknown packet from client. #8084 (Azat Khuzhin)
- 避免空取消引用后“Unknown packet X from server” #8071 (Azat Khuzhin)
- 恢复对所有ICU区域设置的支持，添加对常量表达式应用排序规则的能力，并将语言名称添加到系统。排序规则表。 #8051 (阿利沙平)
- 用于读取的流数 StorageFile 和 StorageHDFS 现在是有限的，以避免超过内存限制。 #7981 (阿利沙平)
- 固定 CHECK TABLE 查询为 *MergeTree 表没有关键. #7979 (阿利沙平)
- 如果没有突变，则从部件名称中删除突变编号。这种删除提高了与旧版本的兼容性。 #8250 (阿利沙平)
- 修复了某些附加部分因data_version大于表突变版本而跳过突变的问题。 #7812 (余志昌)
- 允许在将部件移动到其他设备后使用冗余副本启动服务器。 #7810 (Vladimir Chebotarev)

- 修正了错误 “Sizes of columns doesn't match” 使用聚合函数列时可能会出现。 #7790 (Boris Granveaud)
- 现在在使用WITH TIES和LIMIT BY的情况下，将抛出一个异常。现在可以使用TOP with LIMIT BY。 #7637 (尼基塔 *米哈伊洛夫)
- 修复字典重新加载，如果它有 `invalidate_query`，停止更新，并在以前的更新尝试一些异常。 #8029 (阿利沙平)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

向后不兼容的更改

- 使用列而不是AST来存储标量子查询结果以获得更好的性能。设置 `enable_scalar_subquery_optimization` 在19.17中添加，默认情况下启用。它会导致以下错误 这 在从以前的版本升级到19.17.2或19.17.3期间。默认情况下，19.17.4中禁用此设置，以便可以从19.16及更早版本升级而不会出现错误。 #7392 (阿莫斯鸟)

新功能

- 添加使用DDL查询创建字典的功能。 #7360 (阿利沙平)
- 略眉露>> `bloom_filter` 支持的索引类型 `LowCardinality` 和 `Nullable` #7363 #7561 (尼古拉*科切托夫)
- 添加功能 `isValidJSON` 要检查传递的字符串是否是有效的json。 #5910 #7293 (Vdimir)
- 执行 `arrayCompact` 功能 #7328 (备忘录)
- 创建函数 `hex` 对于十进制数。它的工作原理如下 `hex(reinterpretAsString())`，但不会删除最后的零字节。 #7355 (米哈伊尔*科罗托夫)
- 添加 `arrayFill` 和 `arrayReverseFill` 函数，用数组中其他元素替换它们前面/后面的元素。 #7380 (hcZ)
- 添加 `CRC32IEEE() / CRC64()` 碌莽禄support: #7480 (Azat Khuzhin)
- 执行 `char` 功能类似于一个 mysql #7486 (sundyl)
- 添加 `bitmapTransform` 功能。它将位图中的值数组转换为另一个值数组，结果是一个新的位图 #7598 (余志昌)
- 已实施 `javaHashUTF16LE()` 功能 #7651 (achimbab)
- 添加 `_shard_num` 分布式引擎的虚拟列 #7624 (Azat Khuzhin)

实验特点

- 支持处理器（新的查询执行管道） `MergeTree`. #7181 (尼古拉*科切托夫)

错误修复

- 修复不正确的浮点解析 `Values` #7817 #7870 (tavplubix)
- 修复启用`trace_log`时可能发生的罕见死锁。 #7838 (filimonov)
- 当生成Kafka表时有任何从中选择的Mv时，防止消息重复 #7265 (伊万)
- 支持 `Array(LowCardinality(Nullable(String)))` 在 `IN`. 决定 #7364 #7366 (achimbab)
- 添加处理 `SQL_TINYINT` 和 `SQL_BIGINT`，并修复处理 `SQL_FLOAT` ODBC桥中的数据源类型。 #7491 (Denis Glazachev)
- 修复聚合 (`avg` 和分位数) 在空的十进制列 #7431 (安德烈*科尼亞耶夫)
- 修复 `INSERT` 变成分布 `MATERIALIZED` 列 #7377 (Azat Khuzhin)
- 略眉露>> `MOVE PARTITION` 如果分区的某些部分已经在目标磁盘或卷上，则可以工作 #7434 (Vladimir Chebotarev)
- 修正了在突变过程中无法创建硬链接的错误 `ReplicatedMergeTree` 在多磁盘配置。 #7558 (Vladimir Chebotarev)
- 修复了当整个部分保持不变并且在另一个磁盘上找到最佳空间时，`MergeTree`上出现突变的错误 #7602 (Vladimir Chebotarev)
- 修正错误 `keep_free_space_ratio` 未从磁盘读取配置 #7645 (Vladimir Chebotarev)
- 修正错误与表只包含 `Tuple` 列或具有复杂路径的列。修复 7541. #7545 (阿利沙平)
- 在 `max_memory_usage`限制中不考虑缓冲区引擎的内存 #7552 (Azat Khuzhin)
- 修复最终标记用法 `MergeTree` 表排序 `tuple()`. 在极少数情况下，它可能会导致 `Can't adjust last granule` 选择时出错。 #7639 (安东*波波夫)
- 修复了需要上下文操作（例如json函数）的谓词突变中的错误，这可能会导致崩溃或奇怪的异常。 #7664 (阿利沙平)
- 修复转义的数据库和表名称不匹配 `data/` 和 `shadow/` 目录 #7575 (Alexander Burmak)
- Support duplicated keys in RIGHT|FULL JOINs, e.g. `ON t.x = u.x AND t.x = u.y.` 在这种情况下修复崩溃。 #7586 (Artem Zuikov)
- 修复 `Not found column <expression> in block` 当加入表达式与权利或完全连接。 #7641 (Artem Zuikov)
- 再次尝试修复无限循环 `PrettySpace` 格式 #7591 (Olga Khvostikova)
- 修复bug `concat` 函数时，所有的参数 `FixedString` 同样大小的 #7635 (阿利沙平)

- 在定义S3, URL和HDFS存储时使用1个参数的情况下修复了异常。 #7618 (Vladimir Chebotarev)
- 修复查询视图的InterpreterSelectQuery的范围 #7601 (Azat Khuzhin)

改进

- Nullable ODBC-bridge可识别的列和正确处理的NULL值 #7402 (瓦西里*内姆科夫)
- 以原子方式写入分布式发送的当前批次 #7600 (Azat Khuzhin)
- 如果我们无法在查询中检测到列名称的表，则引发异常。 #7358 (Artem Zuikov)
- 添加 merge_max_block_size 设置为 MergeTreeSettings #7412 (Artem Zuikov)
- 查询与 HAVING 而没有 GROUP BY 假设按常量分组。所以, SELECT 1 HAVING 1 现在返回一个结果。 #7496 (阿莫斯鸟)
- 支持解析 (X,) 作为类似python的元组。 #7501, #7562 (阿莫斯鸟)
- 略眉露>> range 函数行为几乎像pythonic。 #7518 (sundyli)
- 添加 constraints 列到表 system.settings #7553 (维塔利*巴拉诺夫)
- Tcp处理程序的更好的Null格式，以便可以使用 select ignore(<expression>) from table format Null 通过clickhouse-client进行性能测量 #7606 (阿莫斯鸟)
- 查询如 CREATE TABLE ... AS (SELECT (1, 2)) 正确解析 #7542 (hcz)

性能改进

- 改进了对短字符串键的聚合性能。 #6243 (Alexander Kuzmenkov, 阿莫斯鸟)
- 运行另一次语法/表达式分析以在常量谓词折叠后获得潜在的优化。 #7497 (阿莫斯鸟)
- 使用存储元信息来评估琐碎 SELECT count() FROM table; #7510 (阿莫斯鸟, 阿列克谢-米洛维多夫)
- 矢量化处理 arrayReduce 与聚合器类似 addBatch. #7608 (阿莫斯鸟)
- 在性能的小改进 Kafka 消费 #7475 (伊万)

构建/测试/包装改进

- 添加对交叉编译的支持到CPU架构AARCH64。重构打包器脚本。 #7370 #7539 (伊万)
- 在构建软件包时，将darwin-x86_64和linux-aarch64工具链解压缩到已挂载的Docker卷中 #7534 (伊万)
- 更新二进制打包器的Docker映像 #7474 (伊万)
- 修复了MacOS Catalina上的编译错误 #7585 (欧内斯特*波列塔耶夫)
- 查询分析逻辑中的一些重构：将复杂的类拆分为几个简单的类。 #7454 (Artem Zuikov)
- 修复没有子模块的构建 #7295 (proller)
- 更好 add_globs 在CMake文件中 #7418 (阿莫斯鸟)
- 删除硬编码路径 unwind 目标 #7460 (Konstantin Podshumok)
- 允许在没有ssl的情况下使用mysql格式 #7524 (proller)

其他

- 为ClickHouse SQL方言添加了ANTLR4语法 #7595 #7596 (阿列克谢-米洛维多夫)

ClickHouse释放19.16

ClickHouse释放19.16.14.65,2020-03-25

- 修复了多个参数（超过10）的三元逻辑运算批量计算中的错误。 #8718 (亚历山大*卡扎科夫) 这个错误修正是由Altinity的特殊要求回移到版本19.16的。

ClickHouse释放19.16.14.65,2020-03-05

- 修复分布式子查询与旧版本的CH不兼容。修复 #7851 (tabplibix)
- 执行时 CREATE 查询，在存储引擎参数中折叠常量表达式。将空数据库名称替换为当前数据库。修复 #6508, #3492. 还修复检查本地地址 ClickHouseDictionarySource. #9262 (tabplibix)
- 现在背景合并 *MergeTree 表引擎家族更准确地保留存储策略卷顺序。 #8549 (Vladimir Chebotarev)
- 防止丢失数据 Kafka 在极少数情况下，在读取后缀之后但在提交之前发生异常。修复 #9378. 相关: #7175 #9507 (菲利蒙诺夫)
- 修复尝试使用/删除时导致服务器终止的错误 Kafka 使用错误的参数创建的表。修复 #9494. 结合 #9507.

#9513 (菲利蒙诺夫)

- 允许使用 `MaterializedView` 与上面的子查询 `Kafka` 桌子
[#8197 \(filimonov\)](#)

新功能

- 添加 `deduplicate_blocks_in_dependent_materialized_views` 用于控制具有实例化视图的表中幂等插入的行为的选项。这个新功能是由Altinity的特殊要求添加到错误修正版本中的。

[#9070 \(urykhy\)](#)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

向后不兼容的更改

- 为 `count`/`counIf` 添加缺失的验证。

[#7095](#)

[#7298 \(Vdimir\)](#)

- 删除旧版 `asterisk_left_columns_only` 设置（默认情况下禁用）。

[#7335 \(阿尔乔姆](#)

[Zuikov\)](#)

- 模板数据格式的格式字符串现在在文件中指定。

[#7118](#)

[\(tavplubix\)](#)

新功能

- 引入 `uniqCombined64()` 来计算大于 `UINT_MAX` 的基数。

[#7213,](#)

[#7222 \(Azat](#)

[Khuzhin\)](#)

- 支持数组列上的 Bloom filter 索引。

[#6984](#)

[\(achimbab\)](#)

- 添加函数 `getMacro(name)` 返回与相应值的字符串 `<macros>` 从服务器配置。[#7240](#)

[\(阿列克谢-米洛维多夫\)](#)

- 为基于 HTTP 源的字典设置两个配置选项: `credentials` 和 `http-headers`. [#7092 \(纪尧姆](#)

[Tassery\)](#)

- 添加新的 `ProfileEvent Merge` 这计算启动的背景合并的数量。

[#7093 \(米哈伊尔](#)

[科罗托夫\)](#)

- 添加返回完全限定域名的 `fullHostName` 函数。

[#7263](#)

[#7291 \(sundyli\)](#)

- 添加功能 `arraySplit` 和 `arrayReverseSplit` 通过拆分数组 “cut off” 条件。它们在时间序列处理中非常有用。

[#7294 \(hcj\)](#)

- 添加返回 `multiMatch` 函数系列中所有匹配索引的数组的新函数。

[#7299 \(Danila](#)

[库特宁\)](#)

- 添加新的数据库引擎 `Lazy` 即针对存储大量小日志进行了优化桌子 [#7171 \(尼基塔](#)

[Vasilev\)](#)

- 为位图列添加聚合函数 `groupBitmapAnd`, -或-Xor。 [#7109 \(知昌](#)

[阿优\)](#)

- 添加聚合函数组合器-`OrNull`和-`OrDefault`，它们返回 `null` 或默认值时没有任何聚合。

#7331

(hcz)

- 引入支持自定义转义的CustomSeparated数据格式分隔符规则。 #7118 (tavplubix)
- 支持Redis作为外部字典的来源。 #4361 #6962 (comunodi, 安东波波夫)

错误修复

- 修复错误的查询结果，如果它有 WHERE IN (SELECT ...) 部分和 optimize_read_in_order 是使用。 #7371 (安东波波夫)
- 禁用MariaDB身份验证插件，这取决于项目之外的文件。 #7140 (尤里巴拉诺夫)
- 修复异常 Cannot convert column ... because it is constant but values of constants are different in source and result 这可能很少发生，当功能 now(), today(), yesterday(), randConstant() 被使用。 #7156 (尼古拉 Kochetov)
- 修复了使用HTTP保持活动超时而不是TCP保持活动超时的问题。 #7351 (瓦西里 Nemkov)
- 修复了groupBitmapOr中的分段错误（问题 #7109）。 #7289 (知昌 阿优)
- 对于实例化视图，在写入所有数据之后调用kafka的提交。 #7175 (伊万)
- 修复错误 duration_ms 值 system.part_log 桌子 这是十次关闭。 #7172 (弗拉基米尔 Chebotarev)
- 快速修复解决实时查看表中的崩溃并重新启用所有实时查看测试。 #7201 (vzakaznikov)
- 在MergeTree部件的最小/最大索引中正确序列化NULL值。 #7234 (亚历山大 库兹门科夫)
- 不要把虚拟列。创建表时的sql元数据 CREATE TABLE AS. #7183 (伊万)
- 修复分段故障 ATTACH PART 查询。 #7185 (阿利沙平)
- 修复子查询中empty和empty优化给出的某些查询的错误结果 INNER/RIGHT JOIN. #7284 (尼古拉 Kochetov)
- 修复LIVE VIEW getHeader()方法中的AddressSanitizer错误。 #7271 (vzakaznikov)

改进

- 在queue_wait_max_ms 等待发生的情况下添加消息。 #7390 (Azat Khuzhin)
- 制作设置 s3_min_upload_part_size 表级别。

#7059 (弗拉基米尔
Chebotarev)

- 检查Ttl在StorageFactory。 #7304

(sundyli)

- 在部分合并连接（优化）中压缩左侧块。

#7122 (阿尔乔姆
Zuikov)

- 不允许在复制表引擎的突变中使用非确定性函数，因为这可能会在副本之间引入不一致。

#7247 (亚历山大
卡扎科夫)

- 将异常堆栈跟踪转换为字符串时禁用内存跟踪器。它可以防止损失类型的错误消息 `Memory limit exceeded` 在服务器上，这导致了 `Attempt to read after eof` 客户端上的例外。 #7264 (尼古拉*科切托夫)
- 其他格式改进。决定

#6033,

#2633,

#6611,

#6742

#7215

(tavplubix)

- ClickHouse将忽略IN运算符右侧不可转换为左侧的值 side type. Make it work properly for compound types – Array and Tuple.

#7283 (亚历山大
库兹门科夫)

- 支持ASOF加入缺失的不平等。它可以加入小于或等于变体和严格在语法上，ASOF列的变体越来越多。

#7282 (阿尔乔姆
Zuikov)

- 优化部分合并连接。 #7070

(Artem Zuikov)

- 不要在uniqCombined函数中使用超过98K的内存。

#7236,

#7270 (Azat
Khuzhin)

- 在PartialMergeJoin中刷新磁盘上右连接表的部分（如果没有足够的记忆）。需要时加载数据。 #7186
(Artem Zuikov)

性能改进

- 通过避免数据重复加快使用const参数的joinGet。

#7359 (阿莫斯
鸟)

- 如果子查询为空，请提前返回。

#7007 (小路)

- 优化值中SQL表达式的解析。

#6781

(tavplubix)

构建/测试/包装改进

- 禁用交叉编译到Mac OS的一些贡献。

#7101 (伊万)

- 为clickhouse_common_io添加与PocoXML缺少的链接。

#7200 (Azat
Khuzhin)

- 在clickhouse-test中接受多个测试过滤器参数。
[#7226](#) ([亚历山大
库兹门科夫](#))
- 为ARM启用musl和jemalloc。 [#7300](#)
([阿莫斯鸟](#))
- 已添加 `--client-option` 参数 `clickhouse-test` 将其他参数传递给客户端。
[#7277](#) ([尼古拉
Kochetov](#))
- 在rpm软件包升级时保留现有配置。
[#7103](#)
([filimonov](#))
- 修复PVS检测到的错误。 [#7153](#) ([阿尔乔姆
Zuikov](#))
- 修复达尔文的构建。 [#7149](#)
([伊万](#))
- glibc2.29兼容性。 [#7142](#) ([阿莫斯
鸟](#))
- 确保dh_clean不会触及潜在的源文件。
[#7205](#) ([阿莫斯
鸟](#))
- 尝试避免从altinity rpm更新时发生冲突-它有单独打包的配置文件在clickhouse服务器-常见。 [#7073](#)
([filimonov](#))
- 优化一些头文件，以便更快地重建。
[#7212](#),
[#7231](#) ([亚历山大
库兹门科夫](#))
- 添加日期和日期时间的性能测试。 [#7332](#) ([瓦西里
Nemkov](#))
- 修复一些包含非确定性突变的测试。
[#7132](#) ([亚历山大
卡扎科夫](#))
- 添加构建与MemorySanitizer CI。 [#7066](#)
([Alexander Kuzmenkov](#))
- 避免在MetricsTransmitter中使用未初始化的值。
[#7158](#) ([Azat
Khuzhin](#))
- 修复MemorySanitizer发现的字段中的一些问题。
[#7135](#),
[#7179](#) ([亚历山大
库兹门科夫](#)), [#7376](#)
([阿莫斯鸟](#))
- 修复murmurhash32中未定义的行为。 [#7388](#) ([阿莫斯
鸟](#))
- 修复StoragesInfoStream中未定义的行为。 [#7384](#)
([tavplubix](#))
- 固定常量表达式折叠外部数据库引擎（MySQL，ODBC，JDBC）。 在上一页
版本它不适用于多个常量表达式，并且根本不适用于日期，
日期时间和UUID。 这修复 [#7245](#)
[#7252](#)
([阿列克谢-米洛维多夫](#))
- 在访问no_users_thread变量时修复实时查看中的ThreadSanitizer数据竞争错误。
[#7353](#)
([vzakaznikov](#))

- 在 libcommon 中摆脱 malloc 符号
[#7134](#),
[#7065](#) (阿莫斯
鸟)
- 添加全局标志 ENABLE_LIBRARY 以禁用所有库。
[#7063](#)
(aroller)

代码清理

- 概括配置存储库以准备字典的 DDL。 [#7155](#)
(阿利沙平)
- 解析器字典 DDL 没有任何语义。
[#7209](#)
(阿利沙平)
- 将 ParserCreateQuery 拆分为不同的较小的解析器。
[#7253](#)
(阿利沙平)
- 在外部字典附近进行小型重构和重命名。
[#7111](#)
(阿利沙平)
- 重构一些代码以准备基于角色的访问控制。 [#7235](#) (维塔利
巴拉诺夫)
- DatabaseOrdinary 代码中的一些改进。
[#7086](#) (尼基塔
Vasilev)
- 不要在哈希表的 find() 和 emplace() 方法中使用迭代器。
[#7026](#) (亚历山大
库兹门科夫)
- 修正 getMultipleValuesFromConfig 的情况下，当参数根不为空。 [#7374](#)
(米哈伊尔*科罗托夫)
- 删除一些复制粘贴 (TemporaryFile 和 TemporaryFileStream)
[#7166](#) (阿尔乔姆
Zuiakov)
- 改进了代码的可读性一点点 (MergeTreeData::getActiveContainingPart)。
[#7361](#) (弗拉基米尔
Chebotarev)
- 等待使用本地对象的所有计划作业，如果 ThreadPool::schedule(...) 投掷
一个例外 重命名 ThreadPool::schedule(...) 到 ThreadPool::scheduleOrThrowOnError(...) 和
修复注释，使明显的，它可能会抛出。
[#7350](#)
(tavplubix)

ClickHouse 释放 19.15

ClickHouse 释放 19.15.4.10, 2019-10-31

错误修复

- 增加了 sql_tinyint 和 SQL_BIGINT 的处理，并修复了 ODBC 桥中 SQL_FLOAT 数据源类型的处理。
[#7491](#) (Denis Glazachev)
- 允许在移动分区中的目标磁盘或卷上有一些部分。
[#7434](#) (Vladimir Chebotarev)
- 通过 ODBC 桥固定可空列中的 NULL 值。
[#7402](#) (瓦西里*内姆科夫)
- 固定插入到具体化列的分布式非本地节点。
[#7377](#) (Azat Khuzhin)
- 固定函数 getMultipleValuesFromConfig。

- 凸起凸双yeyemuiupic values from coming -

#7374 (米哈伊尔*科罗托夫)

- 修复了使用HTTP保持活动超时而不是TCP保持活动超时的问题。

#7351 (瓦西里*内姆科夫)

- 等待所有作业在异常时完成（修复罕见的段错误）。

#7350 (tavplubix)

- 在插入Kafka表时不要推送MVs。

#7265 (伊万)

- 禁用异常堆栈的内存跟踪器。

#7264 (尼古拉*科切托夫)

- 修复了外部数据库转换查询中的错误代码。

#7252 (阿列克谢-米洛维多夫)

- 避免在MetricsTransmitter中使用未初始化的值。

#7158 (Azat Khuzhin)

- 添加了用于测试的宏的示例配置（阿列克谢-米洛维多夫）

ClickHouse释放19.15.3.6,2019-10-09

错误修复

- 修正了哈希字典中的bad_variant。

(阿利沙平)

- 修复了附加部件查询中分段故障的错误。

(阿利沙平)

- 固定时间计算 MergeTreeData.

(Vladimir Chebotarev)

- 写作完成后明确提交给Kafka。

#7175 (伊万)

- 在MergeTree部件的最小/最大索引中正确序列化NULL值。

#7234 (Alexander Kuzmenkov)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- 分层存储：支持使用MergeTree引擎对表使用多个存储卷。可以将新数据存储在SSD上，并自动将旧数据移动到HDD。（示例）#4918 (Igr) #6489 (阿利沙平)

- 添加表函数 input 用于读取传入的数据 INSERT SELECT 查询。#5450 (palasonic1) #6832 (安东*波波夫)

- 添加一个 sparse_hashed 字典布局，即在功能上等同于 hashed 布局，但更高效的内存。它使用的内存减少了大约两倍，代价是较慢的值检索。#6894 (Azat Khuzhin)

- 实现定义用户列表以访问字典的能力。仅使用当前连接的数据库。#6907 (纪尧姆*塔瑟里)

- 添加 LIMIT 选项 SHOW 查询。#6944 (Philipp Malkovsky)

- 添加 bitmapSubsetLimit(bitmap, range_start, limit) 函数，返回最小的子集 limit 设置中的值不小于 range_start。#6957 (余志昌)

- 添加 bitmapMin 和 bitmapMax 功能。#6970 (余志昌)

- 添加功能 repeat 有关 问题-6648 #6999 (弗林)

实验特点

- 实现（在内存中）不更改当前管道的合并联接变体。结果按合并键进行部分排序。设置 partial_merge_join = 1 要使用此功能。合并联接仍在开发中。#6940 (Artem Zvikov)

- 添加 S3 发动机和表功能。它仍在开发中（还没有身份验证支持）。#5596 (Vladimir Chebotarev)

改进

- 从Kafka读取的每条消息都是以原子方式插入的。这解决了Kafka引擎的几乎所有已知问题。#6950 (伊万)

- 对分布式查询故障转移的改进。缩短恢复时间，也是现在可配置的，可以看出 system.clusters. #6399 (瓦西里*内姆科夫)

- 直接支持枚举的数值 IN 科。#6766 #6941 (dimarub2000)

- 支持（可选，默认情况下禁用）对URL存储进行重定向。#6914 (maqroll)

- 当具有较旧版本的客户端连接到服务器时添加信息消息。 #6893 (Philipp Malkovsky)
- 删除在分布式表中发送数据的最大退避睡眠时间限制 #6895 (Azat Khuzhin)
- 添加将配置文件事件（计数器）与累积值发送到graphite的能力。它可以在启用 `<events_cumulative>` 在服务器 config.xml. #6969 (Azat Khuzhin)
- 添加自动转换类型 T 到 LowCardinality(T) 在类型的列中插入数据 LowCardinality(T) 在本机格式通过HTTP。 #6891 (尼古拉*科切托夫)
- 添加使用功能的能力 hex 不使用 reinterpretAsString 为 Float32, Float64. #7024 (米哈伊尔*科罗托夫)

构建/测试/包装改进

- 将gdb-index添加到带有调试信息的clickhouse二进制文件。这将加快启动时间 gdb. #6947 (阿利沙平)
- 加速deb包装与补丁dpkg-deb它使用 pigz. #6960 (阿利沙平)
- 设置 enable_fuzzing = 1 启用所有项目代码的libfuzzer检测功能。 #7042 (kyprizel)
- 在CI中添加拆分构建烟雾测试。 #7061 (阿利沙平)
- 添加构建与MemorySanitizer CI。 #7066 (Alexander Kuzmenkov)
- 替换 libsparsehash 与 sparsehash-c11 #6965 (Azat Khuzhin)

错误修复

- 修复了大型表上复杂键的索引分析的性能下降。这修复了#6924。 #7075 (阿列克谢-米洛维多夫)
- 修复从Kafka空主题中选择时导致段错误的逻辑错误。 #6909 (伊万)
- 修复过早的MySQL连接关闭 MySQLBlockInputStream.cpp. #6882 (Clément Rodriguez)
- 返回对非常旧的Linux内核的支持 (修复 #6841) #6853 (阿列克谢-米洛维多夫)
- 修复可能的数据丢失 insert select 在输入流中的空块的情况下进行查询。 #6834 #6862 #6911 (尼古拉*科切托夫)
- 修复功能 ArrayEnumerateUniqRanked 在参数中使用空数组 #6928 (proller)
- 使用数组联接和全局子查询修复复杂的查询。 #6934 (伊万)
- 修复 Unknown identifier 按顺序排列和按多个联接分组的错误 #7022 (Artem Zuikov)
- 固定 MSan 执行函数时发出警告 LowCardinality 争论。 #7062 (尼古拉*科切托夫)

向后不兼容的更改

- 更改了位图*聚合函数状态的序列化格式，以提高性能。无法读取以前版本的位图*的序列化状态。 #6908 (余志昌)

ClickHouse释放19.14

ClickHouse释放19.14.7.15,2019-10-02

错误修复

- 此版本还包含19.11.12.69的所有错误修复。
- 修复了19.14和早期版本之间分布式查询的兼容性。这修复 #7068. #7069 (阿列克谢-米洛维多夫)

ClickHouse释放19.14.6.12,2019-09-19

错误修复

- 修复功能 ArrayEnumerateUniqRanked 在参数中使用空数组。 #6928 (proller)
- 修复了查询中的子查询名称 ARRAY JOIN 和 GLOBAL IN subquery 用化名。如果指定了外部表名，请使用子查询别名。 #6934 (伊万)

构建/测试/包装改进

- 修复 拍打 测试 00715_fetch_merged_or_mutated_part_zookeeper 通过将其重写为shell脚本，因为它需要等待突变应用。 #6977 (亚历山大*卡扎科夫)
- 修正了UBSan和MemSan功能失败 groupUniqArray 使用empty数组参数。这是由于放置空 PaddedPODArray 因为没有调用零单元格值的构造函数，所以将其转换为哈希表零单元格。 #6937 (阿莫斯鸟)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- WITH FILL 修饰符 ORDER BY. (继续 #5069) #6610 (安东*波波夫)
- WITH TIES 修饰符 LIMIT. (继续 #5069) #6610 (安东*波波夫)

- 解析无引号 `NULL` 文字为 `NULL` (如果设置 `format_csv_unquoted_null_literal_as_null=1`). 如果此字段的数据类型不可为空，则使用默认值初始化 `null` 字段 (如果设置 `input_format_null_as_default=1`). #5990 #6055 (tavplubix)
- 支持表函数路径中的通配符 `file` 和 `hdfs`. 如果路径包含通配符，则表将为只读。使用示例: `select * from hdfs('hdfs://hdfs1:9000/some_dir/another_dir/*file{0..9}{0..9}')` 和 `select * from file('some_dir/{some_file,another_file,yet_another}.tsv', 'TSV', 'value UInt32')`. #6092 (Olga Khvostikova)
- 新 `system.metric_log` 表存储的值 `system.events` 和 `system.metrics` 具有指定的时间间隔。#6363 #6467 (尼基塔*米哈伊洛夫) #6530 (阿列克谢-米洛维多夫)
- 允许将 ClickHouse 文本日志写入 `system.text_log` 桌子 #6037 #6103 (尼基塔*米哈伊洛夫) #6164 (阿列克谢-米洛维多夫)
- 在堆栈跟踪中显示私有符号 (这是通过解析 ELF 文件的符号表来完成的)。如果存在调试信息，则在堆栈跟踪中添加有关文件和行号的信息。使用程序中存在的索引符号加速符号名称查找。增加了新的 SQL 函数的反省: `demangle` 和 `addressToLine`. 重命名函数 `symbolizeAddress` 到 `addressToSymbol` 为了一致性。功能 `addressToSymbol` 将返回错位的名称出于性能原因，你必须申请 `demangle`. 添加设置 `allow_introspection_functions` 默认情况下，这是关闭的。#6201 (阿列克谢-米洛维多夫)
- 表函数 `values` (名称不区分大小写)。它允许从读取 `VALUES` 建议的名单 #5984. 示例: `SELECT * FROM VALUES('a UInt64, s String', (1, 'one'), (2, 'two'), (3, 'three'))`. #6217. #6209 (dimarub2000)
- 增加了改变存储设置的功能。语法: `ALTER TABLE <table> MODIFY SETTING <setting> = <value>`. #6366 #6669 #6685 (阿利沙平)
- 用于拆卸分离部件的支撑。语法: `ALTER TABLE <table_name> DROP DETACHED PART '<part_id>'`. #6158 (tavplubix)
- 表约束。允许将约束添加到将在插入时检查的表定义。#5273 (格列布*诺维科夫) #6652 (阿列克谢-米洛维多夫)
- 支持级联实例化视图。#6324 (阿莫斯鸟)
- 默认情况下，打开查询探查器以每秒对每个查询执行线程进行一次采样。#6283 (阿列克谢-米洛维多夫)
- 输入格式 ORC. #6454 #6703 (akonyaev90)
- 增加了两个新功能: `sigmoid` 和 `tanh` (这对于机器学习应用程序非常有用)。#6254 (阿列克谢-米洛维多夫)
- 功能 `hasToken(haystack, token)`, `hasTokenCaseInsensitive(haystack, token)` 检查给定的令牌是否在干草堆中。Token 是两个非字母数字 ASCII 字符 (或干草堆的边界) 之间的最大长度子串。Token 必须是常量字符串。由 `tokenbf_v1` 索引专业化支持。#6596, #6662 (瓦西里*内姆科夫)
- 新功能 `neighbor(value, offset[, default_value])`. 允许在一个数据块中的列中达到上一个/下一个值。#5925 (Alex Krash) 6685365ab8c5b74f9650492c88a012596eb1b0c6 341e2e4587a18065c2da1ca888c73389f48ce36c Alexey Milovidov
- 创建了一个函数 `currentUser()`，返回授权用户的登录。添加别名 `user()` 对于与 MySQL 的兼容性。#6470 (Alex Krash)
- 新的聚合函数 `quantilesExactInclusive` 和 `quantilesExactExclusive` 这是在提出 #5885. #6477 (dimarub2000)
- 功能 `bitmapRange(bitmap, range_begin, range_end)` 返回具有指定范围的新集 (不包括 `range_end`). #6314 (余志昌)
- 功能 `geohashesInBox(longitude_min, latitude_min, longitude_max, latitude_max, precision)` 它创建了一系列精确的长串 geohash 盒复盖提供的区域。#6127 (瓦西里*内姆科夫)
- 实现对插入查询的支持 Kafka 桌子 #6012 (伊万)
- 增加了对 `_partition` 和 `_timestamp` 虚拟列到 Kafka 引擎。#6400 (伊万)
- 可以从中删除敏感数据 `query_log`，服务器日志，基于正则表达式的规则的进程列表。#5710 (filimonov)

实验特点

- 输入和输出数据格式 `Template`. 它允许为输入和输出指定自定义格式字符串。#4354 #6727 (tavplubix)
- 执行 `LIVE VIEW` 最初提出的表 #2898，准备 #3925，然后更新 #5541. 看 #5541 详细描述。#5541 (vzakaznikov) #6425 (尼古拉*科切托夫) #6656 (vzakaznikov) 请注意 `LIVE VIEW` 功能可能会在下一个版本中删除。

错误修复

- 此版本还包含 19.13 和 19.11 的所有错误修复。
- 修复表有跳过索引和垂直合并发生时的分段错误。#6723 (阿利沙平)
- 使用非平凡的列默认值修复每列 `TTL`。以前在强制 `TTL` 合并的情况下 `OPTIMIZE ... FINAL` 查询，过期的值被替换为类型默认值，而不是用户指定的列默认值。#6796 (安东*波波夫)
- 修复 kafka 服务器正常重启时的消息重复问题。#6597 (伊万)
- 修正了读取 Kafka 消息时的无限循环。根本不要暂停/恢复订阅消费者-否则在某些情况下可能会无限期暂停。#6354 (伊万)

- 修复 Key expression contains comparison between incompatible types 例外 bitmapContains 功能。 #6136 #6146 #6156 (dimarub2000)
- 修复已启用的段错误 optimize_skip_unused_shards 还丢失了分片钥匙 #6384 (安东*波波夫)
- 修复了可能导致内存损坏的突变中的错误代码。修复了读取地址的段错误 0x14c0 这可能发生由于并发 DROP TABLE 和 SELECT 从 system.parts 或 system.parts_columns. 在准备突变查询时修复了竞争条件。修复了由于 OPTIMIZE 复制的表和并发修改操作，如改变。 #6514 (阿列克谢-米洛维多夫)
- 在MySQL界面中删除了额外的详细日志记录 #6389 (阿列克谢-米洛维多夫)
- 返回解析布尔设置的能力 'true' 和 'false' 在配置文件中。 #6278 (阿利沙平)
- 修复崩溃 quantile 和 median 功能结束 Nullable(Decimal128). #6378 (Artem Zuikov)
- 修正了可能不完整的结果返回 SELECT 查询与 WHERE 主键上的条件包含转换为浮点类型。它是由不正确的单调性检查引起的 toFloat 功能。 #6248 #6374 (dimarub2000)
- 检查 max_expanded_ast_elements 设置为突变。明确突变后 TRUNCATE TABLE. #6205 (张冬)
- 修复使用键列时的联接结果 join_use_nulls. 附加空值而不是列默认值。 #6249 (Artem Zuikov)
- 修正了跳过索引与垂直合并和改变。修复 Bad size of marks file 例外。 #6594 #6713 (阿利沙平)
- 修复罕见的崩溃 ALTER MODIFY COLUMN 和垂直合并，当合并/改变的部分之一是空的 (0行) #6746 #6780 (阿利沙平)
- 修正错误的转换 LowCardinality 类型 AggregateFunctionFactory. 这修复 #6257. #6281 (尼古拉*科切托夫)
- 修复错误的行为和可能的段错误 topK 和 topKWeighted 聚合函数。 #6404 (安东*波波夫)
- 固定周围的不安全代码 getIdentifier 功能。 #6401 #6409 (阿列克谢-米洛维多夫)
- 在MySQL线协议（连接到ClickHouse的形式MySQL客户端时使用）修正了错误。引起的堆缓冲区溢出 PacketPayloadWriteBuffer. #6212 (尤里*巴拉诺夫)
- 固定内存泄漏 bitmapSubsetInRange 功能。 #6819 (余志昌)
- 修复粒度变化后执行突变时的罕见错误。 #6816 (阿利沙平)
- 默认情况下允许包含所有字段的protobuf消息。 #6132 (维塔利*巴拉诺夫)
- 解决错误 nullif 功能，当我们发送 NULL 第二个参数的参数。 #6446 (纪尧姆*塔瑟里)
- 修正了错误的内存分配/解除分配在复杂的键高速缓存字典与字符串字段，导致无限的内存消耗罕见的错误（看起来像内存泄漏）。当字符串大小为8 (8, 16, 32等) 开始的2的幂时，错误会重现。 #6447 (阿利沙平)
- 修复了导致异常的小序列上的大猩猩编码 Cannot write after end of buffer. #6398 #6444 (瓦西里*内姆科夫)
- 允许在连接中使用不可为空的类型 join_use_nulls 已启用。 #6705 (Artem Zuikov)
- 禁用 Poco::AbstractConfiguration 查询中的替换 clickhouse-client. #6706 (阿列克谢-米洛维多夫)
- 避免死锁 REPLACE PARTITION. #6677 (阿列克谢-米洛维多夫)
- 使用 arrayReduce 对于不变的参数可能会导致段错误。 #6242 #6326 (阿列克谢-米洛维多夫)
- 修复可能出现的不一致的部分，如果副本恢复后 DROP PARTITION. #6522 #6523 (tavplubix)
- 固定挂起 JSONExtractRaw 功能。 #6195 #6198 (阿列克谢-米洛维多夫)
- 修正错误跳过索引序列化和聚合与自适应粒度。 #6594. #6748 (阿利沙平)
- 修复 WITH ROLLUP 和 WITH CUBE 修饰符 GROUP BY 具有两级聚合。 #6225 (安东*波波夫)
- 修复编写具有自适应粒度的二级索引标记的错误。 #6126 (阿利沙平)
- 修复服务器启动时的初始化顺序。由于 StorageMergeTree::background_task_handle 在初始化 startup() 该 MergeTreeBlockOutputStream::write() 可以尝试在初始化之前使用它。只需检查它是否被初始化。 #6080 (伊万)
- 从以前的读取操作中清除数据缓冲区，该操作完成时出现错误。 #6026 (尼古拉)
- 修复为复制*MergeTree表创建新副本时启用自适应粒度的错误。 #6394 #6452 (阿利沙平)
- 修复了在服务器启动过程中发生异常的情况下可能发生的崩溃 libunwind 在异常访问未初始化 ThreadStatus 结构。 #6456 (尼基塔*米哈伊洛夫)
- 修复崩溃 yandexConsistentHash 功能。通过模糊测试发现。 #6304 #6305 (阿列克谢-米洛维多夫)
- 修复了服务器过载和全局线程池接近满时挂起查询的可能性。这在具有大量分片（数百个）的集群上发生的机会更高，因为分布式查询为每个分片分配每个连接的线程。例如，如果集群330分片正在处理30个并发分布式查询，则此问题可能再现。此问题会影响从19.2开始的所有版本。 #6301 (阿列克谢-米洛维多夫)
- 的固定逻辑 arrayEnumerateUniqRanked 功能。 #6423 (阿列克谢-米洛维多夫)
- 解码符号表时修复段错误。 #6603 (阿莫斯鸟)
- 在固定不相关的异常转换 LowCardinality(NULLable) to not-NULLable column in case if it doesn't contain Nulls (e.g. in query like SELECT CAST(CAST('Hello' AS LowCardinality(NULLable(String))) AS String). #6094 #6119 (尼古拉*科切托夫)
- 删除描述中的额外引用 system.settings 桌子 #6696 #6699 (阿列克谢-米洛维多夫)
- 避免可能的死锁 TRUNCATE 复制的表。 #6695 (阿列克谢-米洛维多夫)

- 修复读取排序键的顺序。 #6189 (安东*波波夫)
- 修复 ALTER TABLE ... UPDATE 查询表 enable_mixed_granularity_parts=1. #6543 (阿利沙平)
- 修复错误打开 #4405 (自19.4.0)。当我们不查询任何列时，在对MergeTree表的分布式表的查询中复制 (SELECT 1). #6236 (阿利沙平)
- 在有符号类型的整数划分为无符号类型的固定溢出。这种行为与C或C++语言（整数升级规则）完全相同，这可能令人惊讶。请注意，当将大型有符号数字划分为大型无符号数字或反之亦然时，溢出仍然是可能的（但这种情况不太常见）。所有服务器版本都存在此问题。 #6214 #6233 (阿列克谢-米洛维多夫)
- 限制最大睡眠时间限制时 max_execution_speed 或 max_execution_speed_bytes 已设置。修正错误，如 Estimated query execution time (inf seconds) is too long. #5547 #6232 (阿列克谢-米洛维多夫)
- 关于使用固定的问题 MATERIALIZED 列和别名 MaterializedView. #448 #3484 #3450 #2878 #2285 #3796 (阿莫斯鸟) #6316 (阿列克谢-米洛维多夫)
- 修复 FormatFactory 未实现为处理器的输入流的行为。 #6495 (尼古拉*科切托夫)
- 固定错字。 #6631 (Alex Ryndin)
- 错字在错误消息 (是->是)。 #6839 (Denis Zhuravlev)
- 修复了从字符串中解析列列表时的错误，如果类型包含逗号（这个问题与 File, URL, HDFS 储存）#6217. #6209 (dimarub2000)

安全修复

- 此版本还包含19.13和19.11的所有错误安全修复。
- 修复了由于SQL解析器中的堆栈溢出而导致服务器崩溃的制造查询的可能性。修复了合并和分布式表，实例化视图和涉及子查询的行级安全性条件中堆栈溢出的可能性。 #6433 (阿列克谢-米洛维多夫)

改进

- 三元逻辑的正确实现 AND/OR. #6048 (亚历山大*卡扎科夫)
- 现在，值和行与过期的TTL将被删除后 OPTIMIZE ... FINAL query from old parts without TTL infos or with outdated TTL infos, e.g. after ALTER ... MODIFY TTL 查询。添加查询 SYSTEM STOP/START TTL MERGES 要禁止/允许使用TTL分配合并，并在所有合并中过滤过期值。 #6274 (安东*波波夫)
- 可以更改ClickHouse历史文件的位置为客户端使用 CLICKHOUSE_HISTORY_FILE env #6840 (filimonov)
- 删除 dry_run 从标志 InterpreterSelectQuery. ... #6375 (尼古拉*科切托夫)
- 碌莽禄Support: ASOF JOIN 与 ON 科。 #6211 (Artem Zuikov)
- 更好地支持用于突变和复制的跳过索引。支持 MATERIALIZE/CLEAR INDEX ... IN PARTITION 查询。 UPDATE x = x 重新计算使用列的所有索引 x. #5053 (尼基塔*瓦西列夫)
- 允许 ATTACH 实时视图（例如，在服务器启动时），无论 allow_experimental_live_view 设置。 #6754 (阿列克谢-米洛维多夫)
- 对于由查询探查器收集的堆栈跟踪，不包括由查询探查器本身生成的堆栈帧。 #6250 (阿列克谢-米洛维多夫)
- 现在表函数 values, file, url, hdfs 支持别名列。 #6255 (阿列克谢-米洛维多夫)
- 如果抛出异常 config.d 文件没有相应的根元素作为配置文件。 #6123 (dimarub2000)
- 在异常消息中打印额外的信息 no space left on device. #6182, #6252 #6352 (tavplubix)
- 当确定一个碎片 Distributed 要被读取查询复盖的表（用于 optimize_skip_unused_shards =1）ClickHouse现在从两个检查条件 prewhere 和 where select语句的子句。 #6521 (亚历山大*卡扎科夫)
- 已启用 SIMDJSON 对于没有AVX2，但与SSE4.2和PCLMUL指令集的机器。 #6285 #6320 (阿列克谢-米洛维多夫)
- ClickHouse可以在文件系统上工作，而无需 O_DIRECT 支持（如ZFS和Btrfs），无需额外的调整。 #4449 #6730 (阿列克谢-米洛维多夫)
- 支持最终子查询的下推谓词。 #6120 (TCeason) #6162 (阿列克谢-米洛维多夫)
- 更好 JOIN ON 密钥提取 #6131 (Artem Zuikov)
- Updated SIMDJSON. #6285. #6306 (阿列克谢-米洛维多夫)
- 优化最小列的选择 SELECT count() 查询。 #6344 (阿莫斯鸟)
- 已添加 strict 参数 in windowFunnel(). 当 strict 设置，该 windowFunnel() 仅对唯一值应用条件。 #6548 (achimbab)
- 更安全的界面 mysqlxx::Pool. #6150 (avasiliev)
- 执行时选项行大小 --help 选项现在与终端大小对应。 #6590 (dimarub2000)
- 禁用“read in order”优化无键的聚合。 #6599 (安东*波波夫)
- Http状态代码 INCORRECT_DATA 和 TYPE_MISMATCH 错误代码已从默认值更改 500 Internal Server Error 到 400 Bad Request. #6271 (亚历山大*罗丹)
- 从移动连接对象 ExpressionAction 成 AnalyzedJoin. ExpressionAnalyzer 和 ExpressionAction 不知道 Join 不再上课了

它的逻辑被隐藏 AnalyzedJoin 伊菲斯 #6801 (Artem Zuikov)

- 修复了当其中一个分片是localhost但查询通过网络连接发送时可能出现的分布式查询死锁。#6759 (阿列克谢-米洛维多夫)
- 更改多个表的语义 RENAME 为了避免可能的死锁。#6757, #6756 (阿列克谢-米洛维多夫)
- 重写MySQL兼容性服务器以防止在内存中加载完整的数据包有效负载。每个连接的内存消耗减少到大约 2 * DBMS_DEFAULT_BUFFER_SIZE (读/写缓冲区)。#5811 (尤里*巴拉诺夫)
- 将AST别名解释逻辑移出不必了解查询语义的解析器。#6108 (Artem Zuikov)
- 稍微更安全的解析 NamesAndTypesList. #6408, #6410 (阿列克谢-米洛维多夫)
- clickhouse-copier:允许使用 where_condition 从配置 partition_key 查询中用于检查分区存在的别名 (以前它仅用于读取数据查询)。#6577 (proller)
- 在添加可选的消息参数 throwIf. (#5772) #6329 (Vdimir)
- 在发送插入数据时，服务器异常也正在客户端中处理。#5891 #6711 (dimarub2000)
- 添加了指标 DistributedFilesToInsert 这显示了文件系统中选择通过分布式表发送到远程服务器的文件总数。该数字在所有分片之间相加。#6600 (阿列克谢-米洛维多夫)
- 将大部分连接准备逻辑从 ExpressionAction/ExpressionAnalyzer 到 AnalyzedJoin. #6785 (Artem Zuikov)
- 修复曾警告 'lock-order-inversion'. #6740 (瓦西里*内姆科夫)
- 关于缺乏Linux功能的更好的信息消息。记录致命错误 "fatal" 水平，这将使它更容易找到 system.text_log. #6441 (阿列克谢-米洛维多夫)
- 当启用转储临时数据到磁盘，以限制内存使用期间 GROUP BY, ORDER BY，它没有检查可用磁盘空间。修复程序添加新设置 min_free_disk_space，当可用磁盘空间小于阈值时，查询将停止并抛出 ErrorCode::NOT_ENOUGH_SPACE. #6678 (徐伟清) #6691 (阿列克谢-米洛维多夫)
- 通过线程删除递归rwlock。这是没有意义的，因为线程在查询之间重用。SELECT 查询可以在一个线程中获取锁，从另一个线程持有锁并从第一个线程退出。在同一时间，第一个线程可以通过重复使用 DROP 查询。这将导致虚假 "Attempt to acquire exclusive lock recursively" 消息 #6771 (阿列克谢-米洛维多夫)
- 斯普利特 ExpressionAnalyzer.appendJoin(). 准备一个地方 ExpressionAnalyzer 为 MergeJoin. #6524 (Artem Zuikov)
- 已添加 mysql_native_password mysql兼容性服务器的身份验证插件。#6194 (尤里*巴拉诺夫)
- 更少的数量 clock_gettime 调用;调试/发布之间的固定ABI兼容性 Allocator (微不足道的问题)。#6197 (阿列克谢-米洛维多夫)
- 移动 collectUsedColumns 从 ExpressionAnalyzer 到 SyntaxAnalyzer. SyntaxAnalyzer 赖眉露>> required_source_columns 现在本身。#6416 (Artem Zuikov)
- 添加设置 joined_subquery_requires_alias 要求子选择和表函数的别名 FROM that more than one table is present (i.e. queries with JOINs). #6733 (Artem Zuikov)
- 提取物 GetAggregatesVisitor 从类 ExpressionAnalyzer. #6458 (Artem Zuikov)
- system.query_log:更改数据类型 type 列到 Enum. #6265 (尼基塔*米哈伊洛夫)
- 静态链接 sha256_password 身份验证插件。#6512 (尤里*巴拉诺夫)
- 避免对设置的额外依赖 compile 去工作 在以前的版本中，用户可能会得到如下错误 cannot open crt1.o, unable to find library -lc 等。#6309 (阿列克谢-米洛维多夫)
- 对可能来自恶意副本的输入进行更多验证。#6303 (阿列克谢-米洛维多夫)
- 现在 clickhouse-obfuscator 文件是可用的 clickhouse-client 包。在以前的版本中，它可以作为 clickhouse obfuscator (带空格)。#5816 #6609 (dimarub2000)
- 当我们至少有两个查询以不同的顺序读取至少两个表，另一个查询对其中一个表执行DDL操作时，修复了死锁。修复了另一个非常罕见的死锁。#6764 (阿列克谢-米洛维多夫)
- 已添加 os_thread_ids 列到 system.processes 和 system.query_log 为了更好的调试可能性。#6763 (阿列克谢-米洛维多夫)
- 当发生PHP mysqlnd扩展错误的解决方法 sha256_password 用作默认身份验证插件 (在描述 #6031). #6113 (尤里*巴拉诺夫)
- 删除不需要的地方与更改为空列。#6693 (Artem Zuikov)
- 设置默认值 queue_max_wait_ms 为零，因为当前值 (五秒) 是没有意义的。在极少数情况下，此设置有任何用途。添加设置 replace_running_query_max_wait_ms, kafka_max_wait_ms 和 connection_pool_max_wait_ms 用于消除歧义。#6692 (阿列克谢-米洛维多夫)
- 提取物 SelectQueryExpressionAnalyzer 从 ExpressionAnalyzer. 保留最后一个用于非select查询。#6499 (Artem Zuikov)
- 删除重复输入和输出格式。#6239 (尼古拉*科切托夫)

- 允许用户覆盖 `poll_interval` 和 `idle_connection_timeout` 连接设置。#6230 (阿列克谢-米洛维多夫)
- MergeTree 现在有一个额外的选项 `ttl_only_drop_parts` (默认情况下禁用)，以避免部分的部分修剪，以便在部分中的所有行都过期时完全删除它们。#6191 (塞尔吉*弗拉季金)
- 类型检查 `set` 索引函数。如果函数类型错误，则引发异常。这修复了模糊测试与UBSan。#6511 (尼基塔*瓦西列夫)

性能改进

- 优化查询 `ORDER BY expressions` 条款，其中 `expressions` 有重合前缀与排序键 `MergeTree` 桌子 此优化由以下方式控制 `optimize_read_in_order` 设置。#6054 #6629 (安东*波波夫)
- 允许在零件装载和拆卸期间使用多个螺纹。#6372 #6074 #6438 (阿列克谢-米洛维多夫)
- 实现了更新聚合函数状态的批处理变体。这可能导致性能优势。#6435 (阿列克谢-米洛维多夫)
- 使用 `FastOps` 函数库 `exp`, `log`, `sigmoid`, `tanh`. `FastOps` 是迈克尔*帕拉欣 (Yandex 的首席技术官) 的快速矢量数学库。改进的性能 `exp` 和 `log` 功能超过6倍。功能 `exp` 和 `log` 从 `Float32` 参数将返回 `Float32` (在以前的版本中，他们总是返回 `Float64`). 现在 `exp(nan)` 可能会回来 `inf`. 的结果 `exp` 和 `log` 函数可能不是最接近机器可代表的数字到真正的答案。#6254 (阿列克谢-米洛维多夫) 使用 Danila Kutenin 变体使 `fastops` 工作 #6317 (阿列克谢-米洛维多夫)
- 禁用连续密钥优化 `UInt8/16`. #6298 #6701 (akuzm)
- 改进的性能 `simdjson` 库通过摆脱动态分配 `ParsedJson::Iterator`. #6479 (维塔利*巴拉诺夫)
- 预故障页分配内存时 `mmap()`. #6667 (akuzm)
- 修复性能错误 `Decimal` 比较。#6380 (Artem Zuikov)

构建/测试/包装改进

- 删除编译器 (运行时模板实例化)，因为我们已经赢得了它的性能。#6646 (阿列克谢-米洛维多夫)
- 增加了性能测试，以显示 `gcc-9` 以更隔离的方式性能下降。#6302 (阿列克谢-米洛维多夫)
- 添加表功能 `numbers_mt`，这是多线程版本 `numbers`. 使用哈希函数更新性能测试。#6554 (尼古拉*科切托夫)
- 比较模式 `clickhouse-benchmark` #6220 #6343 (dimarub2000)
- 尽最大努力打印堆栈痕迹。还添加了 `SIGPROF` 作为调试信号，打印正在运行的线程的堆栈跟踪。#6529 (阿列克谢-米洛维多夫)
- 每个函数都在自己的文件中，第10部分。#6321 (阿列克谢-米洛维多夫)
- 删除两倍常量 `TABLE_IS_READ_ONLY`. #6566 (filimonov)
- 格式化更改 `StringHashMap` PR #5417. #6700 (akuzm)
- 更好的连接创建子查询 `ExpressionAnalyzer`. #6824 (Artem Zuikov)
- 删除冗余条件 (由 PVS Studio 找到)。#6775 (akuzm)
- 分隔散列表接口 `ReverselIndex`. #6672 (akuzm)
- 重构设置。#6689 (阿利沙平)
- 添加注释 `set` 索引函数。#6319 (尼基塔*瓦西列夫)
- 在 Linux 上的调试版本中增加 OOM 分数。#6152 (akuzm)
- HDFS HA 现在在调试版本中工作。#6650 (徐伟清)
- 添加了一个测试 `transform_query_for_external_database`. #6388 (阿列克谢-米洛维多夫)
- 为 Kafka 表添加多个实例化视图的测试。#6509 (伊万)
- 制定一个更好的构建计划。#6500 (伊万)
- 固定 `test_external_dictionaries` 集成的情况下，它是在非 `root` 用户下执行。#6507 (尼古拉*科切托夫)
- 当写入的数据包的总大小超过 `DBMS_DEFAULT_BUFFER_SIZE`. #6204 (尤里*巴拉诺夫)
- 增加了一个测试 `RENAME` 表竞争条件 #6752 (阿列克谢-米洛维多夫)
- 避免在设置数据竞赛 `KILL QUERY`. #6753 (阿列克谢-米洛维多夫)
- 通过缓存字典添加处理错误的集成测试。#6755 (维塔利*巴拉诺夫)
- 在 Mac OS 上禁用 ELF 对象文件的解析，因为这是没有意义的。#6578 (阿列克谢-米洛维多夫)
- 尝试使更新日志生成器更好。#6327 (阿列克谢-米洛维多夫)
- 添加 `-Wshadow` 切换到海湾合作委员会。#6325 (kreuzerkrieg)
- 删除过时的代码 `mimalloc` 支持。#6715 (阿列克谢-米洛维多夫)
- `zlib-ng` 确定 x86 功能并将此信息保存到全局变量。这是在 `defaltelinit` 调用中完成的，它可以由不同的线程同时进行。为了避免多线程写入，请在库启动时执行此操作。#6141 (akuzm)
- 回归测试一个错误，在连接这是固定的 #5192. #6147 (Bakhtiyor Ruziev)
- 修正 MSan 报告。#6144 (阿列克谢-米洛维多夫)
- 修复 `ttl` 测试。#6782 (安东*波波夫)
- 修正了虚假数据竞赛 `MergeTreeDataPart::is_frozen` 场。#6583 (阿列克谢-米洛维多夫)
- 修正了模糊测试中的超时。在以前的版本中，它设法在查询中找到虚假挂断 `SELECT * FROM`

- numbers_mt(gccMurmurHash('')). #6582 (阿列克谢-米洛维多夫)
- 添加了调试检查 static_cast 列。 #6581 (阿列克谢-米洛维多夫)
- 在官方RPM软件包中支持Oracle Linux。 #6356 #6585 (阿列克谢-米洛维多夫)
- 从更改json perftests once 到 loop 类型。 #6536 (尼古拉*科切托夫)
- odbc-bridge.cpp 定义 main() 所以它不应该被包括在 clickhouse-lib. #6538 (Origej Desh)
- 测试碰撞 FULL|RIGHT JOIN 右表的键中有空值。 #6362 (Artem Zuikov)
- 为了以防万一，增加了对别名扩展限制的测试。 #6442 (阿列克谢-米洛维多夫)
- 从切换 boost::filesystem 到 std::filesystem 在适当的情况下。 #6253 #6385 (阿列克谢-米洛维多夫)
- 将RPM包添加到网站。 #6251 (阿列克谢-米洛维多夫)
- 为固定添加测试 Unknown identifier 例外 IN 科。 #6708 (Artem Zuikov)
- 简化操作 shared_ptr_helper 因为人们面临困难理解它。 #6675 (阿列克谢-米洛维多夫)
- 增加了固定大猩猩和DoubleDelta编解码器的性能测试。 #6179 (瓦西里*内姆科夫)
- 拆分集成测试 test_dictionaries 分成四个单独的测试。 #6776 (维塔利*巴拉诺夫)
- 修复PVS-Studio中的警告 PipelineExecutor. #6777 (尼古拉*科切托夫)
- 允许使用 library 与ASan字典源. #6482 (阿列克谢-米洛维多夫)
- 增加了从Pr列表生成更新日志的选项。 #6350 (阿列克谢-米洛维多夫)
- 锁定 TinyLog 读取时存储。 #6226 (akuzm)
- 检查CI中损坏的符号链接。 #6634 (阿列克谢-米洛维多夫)
- 增加超时时间 “stack overflow” 测试，因为它可能需要很长的时间在调试构建。 #6637 (阿列克谢-米洛维多夫)
- 添加了双空格检查。 #6643 (阿列克谢-米洛维多夫)
- 修复 new/delete 使用消毒剂构建时的内存跟踪。 跟踪不清楚。 它只防止测试中的内存限制异常。 #6450 (Artem Zuikov)
- 启用链接时对未定义符号的检查。 #6453 (伊万)
- 避免重建 hyperscan 每天 #6307 (阿列克谢-米洛维多夫)
- 在固定的瑞银报告 ProtobufWriter. #6163 (阿列克谢-米洛维多夫)
- 不要允许将查询探查器与消毒器一起使用，因为它不兼容。 #6769 (阿列克谢-米洛维多夫)
- 添加测试计时器失败后重新加载字典。 #6114 (维塔利*巴拉诺夫)
- 修复不一致 PipelineExecutor::prepareProcessor 参数类型。 #6494 (尼古拉*科切托夫)
- 添加了对坏Uri的测试。 #6493 (阿列克谢-米洛维多夫)
- 增加了更多的检查 CAST 功能。 这应该获得更多关于模糊测试中分割故障的信息。 #6346 (尼古拉*科切托夫)
- 已添加 gcc-9 支持 docker/builder 本地生成映像的容器。 #6333 (格列布*诺维科夫)
- 测试主键 LowCardinality(String). #5044 #6219 (dimarub2000)
- 修复了缓慢堆栈跟踪打印影响的测试。 #6315 (阿列克谢-米洛维多夫)
- 添加崩溃的测试用例 groupUniqArray 固定在 #6029. #4402 #6129 (akuzm)
- 固定指数突变测试。 #6645 (尼基塔*瓦西列夫)
- 在性能测试中，不要读取我们没有运行的查询的查询日志。 #6427 (akuzm)
- 现在可以使用任何低基数类型创建实例化视图，而不考虑关于可疑低基数类型的设置。 #6428 (Olga Khvostikova)
- 更新的测试 send_logs_level 设置。 #6207 (尼古拉*科切托夫)
- 修复gcc-8.2下的构建。 #6196 (Max Akhmedov)
- 修复构建与内部libc++。 #6724 (伊万)
- 修复共享构建 rdkafka 图书馆 #6101 (伊万)
- 修复Mac OS构建（不完整）。 #6390 (阿列克谢-米洛维多夫) #6429 (alex-zaitsev)
- 修复“splitted” 碌莽祿.拢. #6618 (阿列克谢-米洛维多夫)
- 其他构建修复: #6186 (阿莫斯鸟) #6486 #6348 (vxider) #6744 (伊万) #6016 #6421 #6491 (proller)

向后不兼容的更改

- 删除很少使用的表函数 catBoostPool 和存储 CatBoostPool. 如果您使用了此表功能，请写电子邮件至 clickhouse-feedback@yandex-team.com. 请注意，CatBoost集成仍然存在，并将受到支持。 #6279 (阿列克谢-米洛维多夫)
- 禁用 ANY RIGHT JOIN 和 ANY FULL JOIN 默认情况下。 设置 any_join_distinct_right_table_keys 设置启用它们。 #5126 #6351 (Artem Zuikov)

ClickHouse释放19.13

ClickHouse释放19.13.6.51,2019-10-02

错误修复

- 此版本还包含19.11.12.69的所有错误修复。

ClickHouse释放19.13.5.44,2019-09-20

错误修复

- 此版本还包含19.14.6.12的所有错误修复。
- 修复了在执行时表的可能不一致的状态 `DROP` 在zookeeper无法访问时查询复制的表。 #6045 #6413 (尼基塔*米哈伊洛夫)
- 修复了StorageMerge中的数据竞赛 #6717 (阿列克谢-米洛维多夫)
- 修复查询分析器中引入的错误，从而导致套接字无休止的recv。 #6386 (阿利沙平)
- 修复执行时过多的CPU使用率 `JSONExtractRaw` 函数在一个布尔值。 #6208 (维塔利*巴拉诺夫)
- 在推送到实例化视图时修复回归。 #6415 (伊万)
- 表函数 `url` 该漏洞是否允许攻击者在请求中注入任意HTTP头。这个问题被发现 尼基塔*季霍米罗夫. #6466 (阿列克谢-米洛维多夫)
- 修复无用 `AST` 检查设置索引。 #6510 #6651 (尼基塔*瓦西列夫)
- 修正了解析 `AggregateFunction` 查询中嵌入的值。 #6575 #6773 (余志昌)
- 修正了错误的行为 `trim` 功能家庭。 #6647 (阿列克谢-米洛维多夫)

ClickHouse释放19.13.4.32,2019-09-10

错误修复

- 此版本还包含19.11.9.52和19.11.10.54的所有错误安全修复。
- 固定数据竞赛 `system.parts` 表和 `ALTER` 查询。 #6245 #6513 (阿列克谢-米洛维多夫)
- 修复了从带有`sample`和`prewhere`的空分布式表中读取流中发生的不匹配标题。 #6167 (钱丽祥) #6823 (尼古拉*科切托夫)
- 修复使用时的崩溃 `IN` 子句带有一个元组的子查询。 #6125 #6550 (tavplubix)
- 修复具有相同列名的大小写 `GLOBAL JOIN ON` 科。 #6181 (Artem Zuikov)
- 修复强制转换类型时的崩溃 `Decimal` 这不支持它。抛出异常代替。 #6297 (Artem Zuikov)
- 修复了崩溃 `extractAll()` 功能。 #6644 (Artem Zuikov)
- 查询转换 `MySQL`, `ODBC`, `JDBC` 表函数现在正常工作 `SELECT WHERE` 具有多个查询 `AND` 表达式。 #6381 #6676 (dimarub2000)
- 添加了以前的声明检查MySQL8集成。 #6569 (拉斐尔*大卫*蒂诺科)

安全修复

- 修复解压缩阶段编解码器中的两个漏洞（恶意用户可以制造压缩数据，导致解压缩中的缓冲区溢出）。 #6670 (Artem Zuikov)

碌莽禄,拢,010-68520682\<url>戮漏鹿芦,酶,虏卤赂拢,110102005602

错误修复

- 修复 `ALTER TABLE ... UPDATE` 查询表 `enable_mixed_granularity_parts=1`. #6543 (阿利沙平)
- 在使用`IN`子句时修复带有元组的子查询。 #6125 #6550 (tavplubix)
- 修复了一个问题，即如果一个陈旧的副本变为活动的，它可能仍然有被删除分区的数据部分。 #6522 #6523 (tavplubix)
- 修正了解析CSV的问题 #6426 #6559 (tavplubix)
- 修正了系统中的数据竞赛。部件表和`ALTER`查询。这修复 #6245. #6513 (阿列克谢-米洛维多夫)
- 修复了可能导致内存损坏的突变中的错误代码。修复了读取地址的段错误 `0x14c0` 这可能发生由于并发 `DROP TABLE` 和 `SELECT` 从 `system.parts` 或 `system.parts_columns`. 在准备突变查询时修复了竞争条件。修复了由于 `OPTIMIZE` 复制的表和并发修改操作，如改变。 #6514 (阿列克谢-米洛维多夫)
- 修复后可能的数据丢失 `ALTER DELETE` 查询表跳过索引。 #6224 #6282 (尼基塔*瓦西列夫)

安全修复

- 如果攻击者具有对ZooKeeper的写入访问权限，并且能够从ClickHouse运行的网络中运行可用的自定义服务器，则可以创建自定义构建的恶意服务器，该服务器将充当ClickHouse副本并将其注册到ZooKeeper中。当另一个副本从恶意副本中获取数据部分时，它可以强制clickhouse-server写入文件系统上的任意路径。由Yandex的信息安全团队Eldar

Zaitov发现。 #6247 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682|<url>戮卤箋拢,010-68520682|<url>

新功能

- 查询级别上的采样探查器。示例. #4247 (laplab) #6124 (阿列克谢-米洛维多夫) #6250 #6283 #6386
 - 允许指定列的列表 `COLUMNS('regexp')` 表达的工作原理就像一个更复杂的变体 * 星号 #5951 (mfridental), (阿列克谢-米洛维多夫)
 - `CREATE TABLE AS table_function()` 现在是可能的 #6057 (dimarub2000)
 - 亚当优化随机梯度下降默认情况下使用 `stochasticLinearRegression()` 和 `stochasticLogisticRegression()` 聚合函数，因为它显示了良好的质量，几乎没有进行任何调整。 #6000 (Quid37)
 - Added functions for working with the custom week number #5212 (杨小姐)
 - `RENAME` 查询现在适用于所有存储。 #5953 (伊万)
 - 现在客户端通过设置从服务器接收任何所需级别的日志 `send_logs_level` 无论服务器设置中指定的日志级别如何。 #5964 (尼基塔*米哈伊洛夫)

向后不兼容的更改

- 设置 `input_format_defaults_for_omitted_fields` 默认情况下启用。分布式表中的插入需要此设置在集群上相同（您需要在滚动更新之前设置它）。它允许计算复杂的默认表达式的省略字段 `JSONEachRow` 和 `CSV*` 格式。它应该是预期的行为，但可能导致可忽略不计的性能差异。#6043 (Artem Zuikov), #5625 (akuzm)

实验特点

- 新的查询处理管道。使用 `experimental_use_processors=1` 选项来启用它。用你自己的麻烦。#4914 (尼古拉*科切托夫)

错误修复

- Kafka 集成已在此版本中修复。
 - 固定 DoubleDelta 编码 Int64 对于大 DoubleDelta 值，改进 DoubleDelta 编码为随机数据 Int32. #5998 (瓦西里*内姆科夫)
 - 固定的高估 max_rows_to_read 如果设置 merge_tree_uniform_read_distribution 置为 0。#6019 (阿列克谢-米洛维多夫)

改进

- 如果抛出异常 config.d 文件没有相应的根元素作为配置文件 #6123 (dimarub2000)

性能改进

- 优化 `count()`. 现在它使用最小的列（如果可能的话）。#6028 (阿莫斯鸟)

构建/测试/包装改进

- 在性能测试中报告内存使用情况。#5899 (akuzm)
 - 修复构建与外部 libcxx #6010 (伊万)
 - 修复共享构建 rdafka 图书馆 #6101 (伊万)

ClickHouse 释放19.11

ClickHouse 释放 19.11.13.74, 2019-11-01

错误修复

- 修复了罕见的崩溃 `ALTER MODIFY COLUMN` 当合并/更改部分之一为空（0行）时，垂直合并。#6780 ([阿利沙平](#))
 - 手动更新 `SIMDJSON`。这修复了stderr文件可能泛滥的错误json诊断消息。#7548 ([亚历山大*卡扎科夫](#))
 - 修正错误 `mrk` 突变的文件扩展名 ([阿利沙平](#))

ClickHouse 释放 19.11.12.69, 2019-10-02

错误修复

- 修复了大型表上复杂键的索引分析的性能下降。这修复 #6924. #7075 (阿列克谢-米洛维多夫)

- 使用分页式写入时发送数据时抛光于的SIGSEGV (Failed to send batch: file with index XXXXX is absent).
[#7032 \(Azat Khuzhin\)](#)
- 修复 Unknown identifier 有多个连接。这修复 [#5254](#). [#7022 \(Artem Zuikov\)](#)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

- 修复从Kafka空主题中选择时导致段错误的逻辑错误。[#6902](#) [#6909 \(伊万\)](#)
- 修复功能 `ArrayEnumerateUniqRanked` 在参数中使用空数组。[#6928 \(proller\)](#)

ClickHouse释放19.11.10.54,2019-09-10

错误修复

- 手动存储Kafka消息的偏移量，以便能够一次性为所有分区提交它们。修复潜在的重复“one consumer - many partitions”场景。[#6872 \(伊万\)](#)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682

- 改进缓存字典中的错误处理。[#6737 \(维塔利·巴拉诺夫\)](#)
- 在功能固定错误 `arrayEnumerateUniqRanked`. [#6779 \(proller\)](#)
- 修复 `JSONExtract` 功能，同时提取 `Tuple` 从JSON。[#6718 \(维塔利·巴拉诺夫\)](#)
- 修复后可能的数据丢失 `ALTER DELETE` 查询表跳过索引。[#6224](#) [#6282 \(尼基塔·瓦西列夫\)](#)
- 固定性能测试。[#6392 \(阿列克谢·米洛维多夫\)](#)
- 实木复合地板：修复读取布尔列。[#6579 \(阿列克谢·米洛维多夫\)](#)
- 修正了错误的行为 `nullif` 常量参数的函数。[#6518 \(纪尧姆·塔瑟里\)](#) [#6580 \(阿列克谢·米洛维多夫\)](#)
- 修复kafka服务器正常重启时的消息重复问题。[#6597 \(伊万\)](#)
- 修正了一个问题，当长 `ALTER UPDATE` 或 `ALTER DELETE` 可能会阻止常规合并运行。如果没有足够的可用线程，则防止突变执行。[#6502](#) [#6617 \(tavplubix\)](#)
- 修正了处理错误“timezone”在服务器配置文件中。[#6709 \(阿列克谢·米洛维多夫\)](#)
- 修复卡夫卡测试。[#6805 \(伊万\)](#)

安全修复

- 如果攻击者具有对ZooKeeper的写入访问权限，并且能够从运行ClickHouse的网络中运行可用的自定义服务器，则可以创建自定义构建的恶意服务器，该服务器将充当ClickHouse副本并将其注册到ZooKeeper中。当另一个副本从恶意副本中获取数据部分时，它可以强制clickhouse-server写入文件系统上的任意路径。由Yandex的信息安全团队Eldar Zaitov发现。[#6247 \(阿列克谢·米洛维多夫\)](#)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复 `ALTER TABLE ... UPDATE` 查询表 `enable_mixed_granularity_parts=1`. [#6543 \(阿利沙平\)](#)
- 在使用IN子句时修复带有元组的子查询。[#6125](#) [#6550 \(tavplubix\)](#)
- 修复了一个问题，即如果一个陈旧的副本变为活动的，它可能仍然有被删除分区的数据部分。[#6522](#) [#6523 \(tavplubix\)](#)
- 修正了解析CSV的问题 [#6426](#) [#6559 \(tavplubix\)](#)
- 修正了系统中的数据竞赛。部件表和ALTER查询。这修复 [#6245](#). [#6513 \(阿列克谢·米洛维多夫\)](#)
- 修复了可能导致内存损坏的突变中的错误代码。修复了读取地址的段错误 `0x14c0` 这可能发生由于并发 `DROP TABLE` 和 `SELECT` 从 `system.parts` 或 `system.parts_columns`. 在准备突变查询时修复了竞争条件。修复了由于 `OPTIMIZE` 复制的表和并发修改操作，如改变。[#6514 \(阿列克谢·米洛维多夫\)](#)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- Kafka集成已在此版本中修复。
- 使用时修复段错误 `arrayReduce` 对于不断的参数。[#6326 \(阿列克谢·米洛维多夫\)](#)
- 固定 `toFloat()` 单调性。[#6374 \(dimarub2000\)](#)
- 修复已启用的段错误 `optimize_skip_unused_shards` 还丢失了分片钥匙。[#6384 \(Curtiz\)](#)
- 的固定逻辑 `arrayEnumerateUniqRanked` 功能。[#6423 \(阿列克谢·米洛维多夫\)](#)
- 从MySQL处理程序中删除了额外的详细日志记录。[#6389 \(阿列克谢·米洛维多夫\)](#)

- 修复错误的行为和可能的段错误 `topK` 和 `topKWeighted` 聚合函数。#6404 (Curtiz)
- 不要公开虚拟列 `system.columns` 桌子 这是向后兼容所必需的。#6406 (阿列克谢-米洛维多夫)
- 修复复杂键缓存字典中字符串字段的内存分配错误。#6447 (阿利沙平)
- 修复创建新副本时启用自适应粒度的错误 `Replicated*MergeTree` 桌子 #6452 (阿利沙平)
- 阅读Kafka消息时修复无限循环。#6354 (abyss7)
- 修复了由于SQL解析器中的堆栈溢出和堆栈溢出的可能性而导致服务器崩溃的编造查询的可能性 `Merge` 和 `Distributed` 表 #6433 (阿列克谢-米洛维多夫)
- 在小序列固定大猩猩编码错误。#6444 (Enmk)

改进

- 允许用户复盖 `poll_interval` 和 `idle_connection_timeout` 连接设置。#6230 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682\<url>戮漏鹿芦,酶,虏卤赂拢,110102003042

错误修复

- 修复了服务器超载时挂起查询的可能性。#6301 (阿列克谢-米洛维多夫)
- 修复 `yandexConsistentHash` 函数中的FPE。这修复 #6304. #6126 (阿列克谢-米洛维多夫)
- 修正错误的转换 `LowCardinality` 类型 `AggregateFunctionFactory`. 这修复 #6257. #6281 (尼古拉*科切托夫)
- 修复解析 `bool` 从设置 `true` 和 `false` 配置文件中的字符串。#6278 (阿利沙平)
- 修复查询中不兼容的流头的罕见错误 `Distributed` 桌子结束 `MergeTree` 表时的一部分 `WHERE` 移动到 `PREWHERE`. #6236 (阿利沙平)
- 在有符号类型的整数划分为无符号类型的固定溢出。这修复 #6214. #6233 (阿列克谢-米洛维多夫)

向后不兼容的更改

- Kafka 还是坏了

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复编写具有自适应粒度的二级索引标记的错误。#6126 (阿利沙平)
- 修复 `WITH ROLLUP` 和 `WITH CUBE` 修饰符 `GROUP BY` 具有两级聚合。#6225 (安东*波波夫)
- 固定挂起 `JSONExtractRaw` 功能。固定 #6195 #6198 (阿列克谢-米洛维多夫)
- 修复 `ExternalLoader::reloadOutdated()` 中的段错误。#6082 (维塔利*巴拉诺夫)
- 修复了服务器可能关闭倾听套接字但不关闭并继续提供剩余查询的情况。您最终可能会有两个正在运行的clickhouse 服务器进程。有时，服务器可能会返回错误 `bad_function_call` 对于剩余的查询。#6231 (阿列克谢-米洛维多夫)
- 修复了通过ODBC，MySQL，ClickHouse和HTTP初始加载外部字典的更新字段无用和不正确的条件。这修复 #6069 #6083 (阿列克谢-米洛维多夫)
- 在固定不相关的异常转换 `LowCardinality(Nullable)` to not-Nullable column in case if it doesn't contain Nulls (e.g. in query like `SELECT CAST(CAST('Hello' AS LowCardinality(Nullable(String))) AS String)`). #6094 #6119 (尼古拉*科切托夫)
- 修复非确定性结果 “`uniq`” 在极少数情况下聚合函数。该错误存在于所有ClickHouse版本。#6058 (阿列克谢-米洛维多夫)
- `Segfault` 当我们在函数上设置了一点点太高的CIDR IPv6CIDRToRange. #6068 (纪尧姆*塔瑟里)
- 修复了服务器从许多不同上下文中抛出许多异常时的小内存泄漏。#6144 (阿列克谢-米洛维多夫)
- 修复消费者在订阅之前暂停而之后未恢复的情况。#6075 (伊万) 请注意，卡夫卡在这个版本中被打破。
- 从以前的读取操作中清除Kafka数据缓冲区，并且完成了错误操作 #6026 (尼古拉) 请注意，卡夫卡在这个版本中被打破。
- 由于 `StorageMergeTree::background_task_handle` 在初始化 `startup()` 该 `MergeTreeBlockOutputStream::write()` 可以尝试在初始化之前使用它。只需检查它是否被初始化。#6080 (伊万)

构建/测试/包装改进

- 新增官方 `rpm` 包. #5740 (proller) (阿利沙平)
- 添加构建能力 `.rpm` 和 `.tgz` 包 `packager` 脚本 #5769 (阿利沙平)
- 修复了“Arcadia”构建系统。#6223 (proller)

向后不兼容的更改

- Kafka 在这个版本中被打破。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- 增加了对准备好的语句的支持。 #5331 (亚历山大) #5630 (阿列克谢-米洛维多夫)
- DoubleDelta 和 Gorilla 列编解 ecs #5600 (瓦西里*内姆科夫)
- 已添加 os_thread_priority 设置，允许控制 “nice” 操作系统用于调整动态调度优先级的查询处理线程的值。它需要 CAP_SYS_NICE 能力的工作。这实现了 #5858 #5909 (阿列克谢-米洛维多夫)
- 执行 _topic, _offset, _key kafka 引擎的列 #5382 (伊万) 请注意，卡夫卡在这个版本中被打破。
- 添加聚合函数组合 -Resample #5590 (hcz)
- 聚合函数 groupArrayMovingSum(win_size)(x) 和 groupArrayMovingAvg(win_size)(x)，计算移动和/平均有或没有窗口大小限制。 #5595 (inv2004)
- 添加synonim arrayFlatten \<-> flatten #5764 (hcz)
- Intergate H3 功能 geoToH3 从尤伯杯。#4724 (Remen Ivan) #5805 (阿列克谢-米洛维多夫)

错误修复

- 使用异步更新实现DNS缓存。单独的线程解析所有主机并更新dns缓存（设置 dns_cache_update_period）。当主机的ip频繁更改时，它应该有所帮助。 #5857 (安东*波波夫)
- 修复段错误 Delta 影响值小于32位大小的列的编解ec。该错误导致随机内存损坏。 #5786 (阿利沙平)
- 修复ttl合并中的段错误与块中的非物理列。 #5819 (安东*波波夫)
- 修复在检查部分罕见的错误 LowCardinality 列。前情提要 checkDataPart 总是失败的一部分 LowCardinality 列。 #5832 (阿利沙平)
- 避免在服务器线程池已满时挂起连接。它是从连接重要 remote 当连接超时时，表函数或连接到没有副本的分片。这修复 #5878 #5881 (阿列克谢-米洛维多夫)
- 支持常量参数 evalMLModel 功能。这修复 #5817 #5820 (阿列克谢-米洛维多夫)
- 修复了ClickHouse将默认时区确定为 UCT 而不是 UTC。这修复 #5804. #5828 (阿列克谢-米洛维多夫)
- 固定缓冲区下溢 visitParamExtractRaw。这修复 #5901 #5902 (阿列克谢-米洛维多夫)
- 现在分发 DROP/ALTER/TRUNCATE/OPTIMIZE ON CLUSTER 查询将直接在leader副本上执行。 #5757 (阿利沙平)
- 修复 coalesce 为 ColumnConst 与 ColumnNullable + 相关变化。#5755 (Artem Zuikov)
- 修复 ReadBufferFromKafkaConsumer 所以它不断阅读新的消息后 commit() 即使它之前停滞不前 #5852 (伊万)
- 修复 FULL 和 RIGHT 加入时加入结果 Nullable 键在右表。#5859 (Artem Zuikov)
- 可能修复低优先级查询的无限休眠。#5842 (阿列克谢-米洛维多夫)
- 修复争用条件，这导致某些查询可能不会出现在query_log后 SYSTEM FLUSH LOGS 查询。#5456 #5685 (安东*波波夫)
- 固定 heap-use-after-free 由手表引起的ClusterCopier中的警告尝试使用已经删除的复印机对象。#5871 (尼古拉*科切托夫)
- 修复错误 StringRef 由一些实现返回的指针 IColumn::deserializeAndInsertFromArena。这个错误只影响单元测试。 #5973 (尼古拉*科切托夫)
- 防止源数组和中间数组连接掩蔽相同名称列的列。#5941 (Artem Zuikov)
- 修复插入并选择查询MySQL引擎与MySQL样式标识符引用。#5704 (张冬)
- 现在 CHECK TABLE 查询可以与MergeTree引擎系列一起使用。它返回检查状态和消息，如果任何为每个部分（或文件在simplifier引擎的情况下）。此外，修复获取损坏部分的错误。#5865 (阿利沙平)
- 修复SPLIT_SHARED_LIBRARY运行时 #5793 (Danila Kutenin)
- 固定时区初始化时 /etc/localtime 是一个相对的符号链接，如 ./usr/share/zoneinfo/Europe/Moscow #5922 (阿列克谢-米洛维多夫)
- clickhouse复印机：修复使用-关机后免费 #5752 (proller)
- 更新 simdjson。修复了一些无效的零字节Json成功解析的问题。#5938 (阿列克谢-米洛维多夫)
- 修复系统日志的关机 #5802 (安东*波波夫)
- 修复当invalidate_query中的条件取决于字典时挂起。#6011 (维塔利*巴拉诺夫)

改进

- 允许群集配置中的无法解析的地址。它们将被视为不可用，并尝试在每次连接尝试时解决。这对Kubernetes特别有用。这修复 #5714 #5924 (阿列克谢-米洛维多夫)
- 关闭空闲TCP连接（默认情况下为一小时超时）。这对于每台服务哭卜且有多个分布式表的大型集群尤其重要，因为每

所有服务器都可能保留与其他服务器的连接池，并且在高峰查询并发之后，连接将停。这修复 #5879 #5880 (阿列克谢-米洛维多夫)

- 更好的质量 `topK` 功能。如果新元素具有更大的权重，则更改了 `SavingSpace set` 行为以删除最后一个元素。#5833 #5850 (纪尧姆*塔瑟里)
- 与域一起使用的URL函数现在可以在没有方案的情况下适用于不完整的Url #5725 (阿利沙平)
- 校验和添加到 `system.parts_columns` 桌子 #5874 (尼基塔*米哈伊洛夫)
- 已添加 `Enum` 数据类型作为synonym `Enum8` 或 `Enum16`. #5886 (dimarub2000)
- 全位转置变种 `T64` 编解ec 可能会导致更好的压缩 `zstd`. #5742 (Artem Zuikov)
- 条件 `startsWith` 函数现在可以使用主键。这修复 #5310 和 #5882 #5919 (dimarub2000)
- 允许使用 `clickhouse-copier` 通过允许空数据库名称来实现具有交叉复制的群集拓扑。#5745 (纳瓦托洛梅)
- 使用 `UTC` 作为系统上的默认时区，而不 `tzdata` (e.g. bare Docker container). Before this patch, error message `Could not determine local time zone` 被打印并且服务器或客户机拒绝启动。#5827 (阿列克谢-米洛维多夫)
- 返回对函数中浮点参数的支持 `quantileTiming` 为了向后兼容性。#5911 (阿列克谢-米洛维多夫)
- 在错误消息中显示哪个表缺少列。#5768 (伊万)
- 不允许不同用户使用相同的`query_id`运行查询 #5430 (proller)
- 用于向Graphite发送指标的更强大的代码。它甚至可以在长时间的多重工作 `RENAME TABLE` 操作。#5875 (阿列克谢-米洛维多夫)
- 当ThreadPool无法计划执行任务时，将显示更多信息错误消息。这修复 #5305 #5801 (阿列克谢-米洛维多夫)
- 反转ngramSearch更直观 #5807 (Danila Kutenin)
- 在HDFS引擎生成器中添加用户解析 #5946 (akonyaev90)
- 更新默认值 `max_ast_elements` parameter #5933 (Artem Konovalov)
- 增加了过时设置的概念。过时的设置 `allow_experimental_low_cardinality_type` 可以没有效果使用。

0f15c01c6802f7ce1a1494c12c846be8c98944cd Alexey Milovidov

性能改进

- 增加从合并表中选择的流数量，以便更均匀地分布线程。添加设置 `max_streams_multiplier_for_merge_tables`. 这修复 #5797 #5915 (阿列克谢-米洛维多夫)

构建/测试/包装改进

- 为与不同版本的clickhouse的客户端-服务器交互添加向后兼容性测试。#5868 (阿利沙平)
- 每个提交和拉取请求中的测试覆盖率信息。#5896 (阿利沙平)
- 与address sanitizer合作，支持我们的自定义分alloc (`Arena` 和 `ArenaWithFreeLists`) 为了更好地调试“use-after-free”错误。#5728 (akuzm)
- 切换到 `LLVM libunwind` 实现 用于C++异常处理和堆栈跟踪打印 #4828 (尼基塔*拉普科夫)
- 添加来自-Weverything的两个警告 #5923 (阿列克谢-米洛维多夫)
- 允许用内存消毒剂建立ClickHouse。#3949 (阿列克谢-米洛维多夫)
- 关于固定的ubsan报告 `bitTest` 在模糊测试功能。#5943 (阿列克谢-米洛维多夫)
- Docker：增加了初始化需要身份验证的ClickHouse实例的可能性。#5727 (科尔维亚科夫*安德烈)
- 将librdkafka更新到版本1.1.0 #5872 (伊万)
- 为集成测试添加全局超时，并在测试代码中禁用其中一些。#5741 (阿利沙平)
- 修复一些ThreadSanitizer故障。#5854 (akuzm)
- 该 `--no-undefined` 选项强制链接器在链接时检查所有外部名称是否存在。在拆分构建模式下跟踪库之间的依赖关系非常有用。#5855 (伊万)
- 增加了性能测试 #5797 #5914 (阿列克谢-米洛维多夫)
- 与gcc-7固定兼容性。#5840 (阿列克谢-米洛维多夫)
- 增加了对gcc-9的支持。这修复 #5717 #5774 (阿列克谢-米洛维多夫)
- 修复了libunwind链接不正确时的错误。#5948 (阿列克谢-米洛维多夫)
- 修复了PVS-Studio发现的一些警告。#5921 (阿列克谢-米洛维多夫)
- 增加了初始支持 `clang-tidy` 静态分析仪。#5806 (阿列克谢-米洛维多夫)
- 转换BSD/Linux endian宏(‘be64toh’ 和 ‘htobe64’) 到Mac OS x当量 #5785 (傅辰)
- 改进的集成测试指南. #5796 (Vladimir Chebotarev)
- 在macosx+gcc9修复构建 #5822 (filimonov)
- 修复难以识别的错字：aggreAGte->aggregate。#5753 (akuzm)
- 修复freebsd构建 #5760 (proller)

- 添加链接到实验YouTube频道的网站 #5845 (伊万*布林科夫)
- CMake：为复盖率标志添加选项：WITH_COVERAGE #5776 (proller)
- 修复一些内联PODArray的初始大小。 #5787 (akuzm)
- clickhouse服务器.postinst：修复centos6的操作系统检测 #5788 (proller)
- 添加Arch linux软件包生成。 #5719 (Vladimir Chebotarev)
- 拆分常见/配置.h by libs(dbms) #5715 (proller)
- 修复了“Arcadia”构建平台 #5795 (proller)
- 修复了非常规构建 (gcc9，没有子模块) #5792 (proller)
- 在unalignedStore中需要显式类型，因为它被证明容易出现错误 #5791 (akuzm)
- 修复MacOS构建 #5830 (filimonov)
- 关于具有更大数据集的新JIT功能的性能测试，请参阅此处 #5263 #5887 (纪尧姆*塔瑟里)
- 在压力测试中运行有状态测试 12693e568722f11e19859742f56428455501fd2a (阿利沙平)

向后不兼容的更改

- Kafka 在这个版本中被打破。
- 启用 adaptive_index_granularity =10mb默认为新 MergeTree 桌子 如果您在19.11+版本上创建了新的MergeTree 表，则不可能降级到19.6之前的版本。 #5628 (阿利沙平)
- 删除了Yandex使用的过时无证嵌入式字典。梅特里卡 功能 OSIn, SEIn, OSToRoot, SEToRoot, OSHierarchy, SEHierarchy 不再可用。如果您正在使用这些功能，请写电子邮件至clickhouse-feedback@yandex-team.com
注：在最后时刻，我们决定保持这些功能一段时间。 #5780 (阿列克谢-米洛维多夫)

ClickHouse释放19.10

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url> 新功能

- 添加新列编解ec: T64. 为 (U) IntX/EnumX/Data (时间) /DecimalX列制作。它应该适用于具有常量或小范围值的列。编解码器本身允许放大或缩小数据类型而无需重新压缩。 #5557 (Artem Zuikov)
- 添加数据库引擎 MySQL 允许查看远程MySQL服务器中的所有表 #5599 (张冬)
- bitmapContains 执行。这是2倍的速度比 bitmapHasAny 如果第二个位图包含一个元素。 #5535 (余志昌)
- 支持 crc32 功能 (与MySQL或PHP中的行为完全相同)。如果您需要散列函数，请不要使用它。 #5661 (Remen Ivan)
- 已实施 SYSTEM START/STOP DISTRIBUTED SENDS 查询控制异步插入到 Distributed 桌子 #4935 (张冬)

错误修复

- 在执行突变时忽略查询执行限制和合并限制的最大部件大小。 #5659 (安东*波波夫)
- 修复可能导致重复数据删除正常块 (极其罕见) 和插入重复块 (更常见) 的错误。 #5549 (阿利沙平)
- 功能修复 arrayEnumerateUniqRanked 对于具有空数组的参数 #5559 (proller)
- 不要在没有轮询任何消息的情况下订阅Kafka主题。 #5698 (伊万)
- 使设置 join_use_nulls 对于不能在Nullable内的类型不起作用 #5700 (Olga Khvostikova)
- 固定 Incorrect size of index granularity 错误 #5720 (coraxster)
- 修正浮动到十进制转换溢出 #5607 (coraxster)
- 冲洗缓冲区时 WriteBufferFromHDFS的析构函数被调用。这修复了写入 HDFS. #5684 (新东鹏)

改进

- 对待空单元格 CSV 作为默认值时的设置 input_format_defaults_for_omitted_fields 被启用。 #5625 (akuzm)
- 外部字典的非阻塞加载。 #5567 (维塔利*巴拉诺夫)
- 可以根据设置动态更改已建立的连接的网络超时。 #4558 (Konstantin Podshumok)
- 使用“public_suffix_list”对于功能 firstSignificantSubdomain, cutToFirstSignificantSubdomain. 它使用一个完美的哈希表生成 gperf 从文件生成的列表：https://publicsuffix.org/list/public_suffix_list.dat (例如，现在我们认识到域 ac.uk 作为非显着)。 #5030 (纪尧姆*塔瑟里)
- 通过 IPv6 系统表中的数据类型;统一客户端信息列 system.processes 和 system.query_log #5640 (阿列克谢-米洛维多夫)
- 使用会话与MySQL兼容性协议的连接。 #5476 #5646 (尤里*巴拉诺夫)
- 支持更多 ALTER 查询 ON CLUSTER. #5593 #5613 (sundyli)

- 禁用 Support: <logger> 第1下 clickhouse-local 配置文件。 #5540 (proller)
- 允许运行查询 remote 表函数 clickhouse-local #5627 (proller)

性能改进

- 添加在MergeTree列末尾写最后标记的可能性。它允许避免对超出表数据范围的键进行无用的读取。仅当使用自适应索引粒度时才启用此功能。 #5624 (阿利沙平)
- 通过减少非常慢的文件系统上的MergeTree表的性能 stat syscalls #5648 (阿列克谢-米洛维多夫)
- 修复了从版本19.6中引入的MergeTree表读取时的性能下降。修复#5631。 #5633 (阿列克谢-米洛维多夫)

构建/测试/包装改进

- 已实施 TestKeeper 作为用于测试的ZooKeeper接口的实现 #5643 (阿列克谢-米洛维多夫) (levushkin aleksej)
- 从现在起 .sql 测试可以通过服务器隔离，并行运行，并使用随机数据库。它允许更快地运行它们，使用自定义服务器配置添加新的测试，并确保不同的测试不会相互影响。 #5554 (伊万)
- 删除 <name> 和 <metrics> 从性能测试 #5672 (Olga Khvostikova)
- 固定 “select_format” 性能测试 Pretty 格式 #5642 (阿列克谢-米洛维多夫)

ClickHouse 释放19.9

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复增量编解码器中的段错误，这会影响值小于32位大小的列。该错误导致随机内存损坏。 #5786 (阿利沙平)
- 修复在检查部分低速率列中罕见的错误。 #5832 (阿利沙平)
- 修复ttl合并中的段错误与块中的非物理列。 #5819 (安东*波波夫)
- 修复低优先级查询的潜在无限休眠。 #5842 (阿列克谢-米洛维多夫)
- 修复ClickHouse如何将默认时区确定为UCT而不是UTC。 #5828 (阿列克谢-米洛维多夫)
- 修复在领导者副本之前的跟随者副本上执行分布式删除/更改/截断/优化集群查询的错误。现在他们将直接在领导者副本上执行。 #5757 (阿利沙平)
- 修复了系统刷新日志查询后某些查询可能不会立即出现在query_log中的竞争条件。 #5685 (安东*波波夫)
- 增加了对常量参数的缺失支持 evalMLModel 功能。 #5820 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- 打印有关冷冻部件的信息 system.parts 桌子 #5471 (proller)
- 在clickhouse上询问客户端密码-如果未在参数中设置，则在tty上启动客户端 #5092 (proller)
- 执行 dictGet 和 dictGetOrDefault 十进制类型的函数。 #5394 (Artem Zuikov)

改进

- Debian的初始化：添加服务停止超时 #5522 (proller)
- 默认情况下添加禁止设置，以创建具有可疑类型的表格 #5448 (Olga Khvostikova)
- 当不用作函数中的状态时，回归函数返回模型权重 evalMLMethod. #5411 (Quid37)
- 重命名和改进回归方法。 #5492 (Quid37)
- 更清晰的字符串搜索界面。 #5586 (Danila Kutenin)

错误修复

- 修复Kafka中潜在的数据丢失 #5445 (伊万)
- 修复潜在的无限循环 PrettySpace 使用零列调用时的格式 #5560 (Olga Khvostikova)
- 修正了线性模型中的UInt32溢出错误。允许对非常量模型参数的eval ML模型。 #5516 (尼古拉*科切托夫)
- ALTER TABLE ... DROP INDEX IF EXISTS ... 如果提供的索引不存在，则不应引发异常 #5524 (格列布*诺维科夫)
- 修复段错误 bitmapHasAny 在标量子查询中 #5528 (余志昌)
- 修复了复制连接池不重试解析主机时的错误，即使删除了DNS缓存。 #5534 (阿利沙平)
- 固定 ALTER ... MODIFY TTL 在ReplicatedMergeTree上。 #5539 (安东*波波夫)
- 修复插入到具体化列的分布式表中 #5429 (Azat Khuzhin)
- 修复截断联接存储时的错误alloc #5437 (TCeason)
- 在最近版本的包tzdata中，现在有些文件是符号链接。当前用于检测默认时区的机制被打破，并为某些时区提供错误的

名称。现在至少我们强制时区名称到TZ的内容，如果提供。[#5443 \(伊万\)](#)

- 修复一些极为罕见的情况下，MultiVolnitsky搜索器时，在总和恒定针至少16KB长。该算法错过或复盖以前的结果，这可能导致错误的结果 `multiSearchAny`. [#5588 \(Danila Kutenin\)](#)
- 修复ExternalData请求的设置无法使用ClickHouse设置时的问题。此外，现在，设置 `date_time_input_format` 和 `low_cardinality_allow_in_native_format` 由于名称的歧义，无法使用（在外部数据中，它可以解释为表格式，在查询中它可以是一个设置）。[#5455 \(Danila Kutenin\)](#)
- 修复只从FS中删除部件而不从Zookeeper中删除部件时的错误。[#5520 \(阿利沙平\)](#)
- 从MySQL协议中删除调试日志记录 [#5478 \(阿列克谢-米洛维多夫\)](#)
- 在DDL查询处理过程中跳过ZNONODE [#5489 \(Azat Khuzhin\)](#)
- 修复混合 UNION ALL 结果列类型。有些情况下，结果列的数据和列类型不一致。[#5503 \(Artem Zuikov\)](#)
- 在错误的整数上抛出异常 `dictGetT` 功能，而不是崩溃。[#5446 \(Artem Zuikov\)](#)
- 修复散列字典中错误的element_count和load_factor `system.dictionaries` 桌子 [#5440 \(Azat Khuzhin\)](#)

构建/测试/包装改进

- 固定构建没有 `Brotli` HTTP压缩支持 (`ENABLE_BROTLI=OFF` cmake变量) 。[#5521 \(Anton Yuzhaninov\)](#)
- 包括ro嗦。`h`为ro嗦/咆哮。`h` [#5523 \(Origej Desh\)](#)
- 修复超扫描中的gcc9警告（#行指令是邪恶的！）[#5546 \(Danila Kutenin\)](#)
- 使用gcc-9编译时修复所有警告。修复一些contrib问题。修复gcc9ICE并将其提交给bugzilla。[#5498 \(Danila Kutenin\)](#)
- 与lld固定链接 [#5477 \(阿列克谢-米洛维多夫\)](#)
- 删除字典中未使用的专业化 [#5452 \(Artem Zuikov\)](#)
- 针对不同类型的文件进行格式化和解析表的改进性能测试 [#5497 \(Olga Khvostikova\)](#)
- 修复并行测试运行 [#5506 \(proller\)](#)
- Docker：使用clickhouse-test中的configs [#5531 \(proller\)](#)
- 修复编译为FreeBSD [#5447 \(proller\)](#)
- 升级提升到1.70 [#5570 \(proller\)](#)
- 修复构建clickhouse作为子模块 [#5574 \(proller\)](#)
- 改进JSONExtract性能测试 [#5444 \(维塔利*巴拉诺夫\)](#)

ClickHouse释放19.8

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- 添加了与JSON一起使用的函数 [#4686 \(hczi\) #5124. \(维塔利*巴拉诺夫\)](#)
- 添加一个函数basename，具有类似于basename函数的行为，它存在于许多语言中 (`os.path.basename` 在python中, `basename` in PHP, etc...). Work with both an UNIX-like path or a Windows path.[#5136 \(纪尧姆*塔瑟里\)](#)
- 已添加 `LIMIT n, m BY` 或 `LIMIT m OFFSET n BY` 为LIMIT BY子句设置n偏移量的语法。[#5138 \(安东*波波夫\)](#)
- 增加了新的数据类型 `SimpleAggregateFunction`，它允许在一个具有光聚集的列 `AggregatingMergeTree`. 这只能用于简单的功能，如 `any, anyLast, sum, min, max.` [#4629 \(Boris Granveaud\)](#)
- 增加了对函数中非常量参数的支持 `ngramDistance` [#5198 \(Danila Kutenin\)](#)
- 新增功能 `skewPop`, `skewSamp`, `kurtPop` 和 `kurtSamp` 分别计算序列偏度、样本偏度、峰度和样本峰度。[#5200 \(hczi\)](#)
- 支持重命名操作 `MaterializeView` 存储。[#5209 \(纪尧姆*塔瑟里\)](#)
- 添加了允许使用MySQL客户端连接到ClickHouse的服务器。[#4715 \(尤里*巴拉诺夫\)](#)
- 添加 `toDecimal*OrZero` 和 `toDecimal*OrNull` 功能。[#5291 \(Artem Zuikov\)](#)
- 支持函数中的十进制类型: `quantile`, `quantiles`, `median`, `quantileExactWeighted`, `quantilesExactWeighted` 媒体加权。[#5304 \(Artem Zuikov\)](#)
- 已添加 `toValidUTF8` function, which replaces all invalid UTF-8 characters by replacement character ♦ (U+FFFD). [#5322 \(Danila Kutenin\)](#)
- 已添加 `format` 功能。使用参数中列出的字符串格式化常量模式（简化的Python格式模式）。[#5330 \(Danila Kutenin\)](#)
- 已添加 `system.detached_parts` 表包含有关分离部分的信息 `MergeTree` 桌子 [#5353 \(akuzm\)](#)
- 已添加 `ngramSearch` 函数来计算针和大海捞针之间的非对称差异。[#5418#5422 \(Danila Kutenin\)](#)
- 使用聚合函数接口实现基本的机器学习方法（随机线性回归和逻辑回归）。有不同的策略，用于更新模型权重（简单梯度下降，岭回归，泊松回归等）。江士林白市川上小的小扯头。[#4043 \(Ovidiu\)](#)

反向操作，例如插入，注册键或插入）。还支持日志插入小时小批处理。#4945 (Quicu51)

- 执行 `geohashEncode` 和 `geohashDecode` 功能。#5003 (瓦西里*内姆科夫)
- 添加聚合功能 `timeSeriesGroupSum`，从而可以聚合不同的时间序列，即采样时间戳不对齐。它将在两个采样时间戳之间使用线性插值，然后将时间序列和在一起。添加聚合功能 `timeSeriesGroupRateSum`，它计算时间序列的速率，然后将速率总和在一起。#4542 (刘杨宽)
- 新增功能 `IPv4CIDRtoIPv4Range` 和 `IPv6CIDRtoIPv6Range` 使用CIDR计算子网中IP的下限和上限。#5095 (纪尧姆*塔瑟里)
- 添加一个X-ClickHouse-Summary头，当我们发送查询使用HTTP启用设置 `send_progress_in_http_headers`。返回X-ClickHouse-Progress的常用信息，以及其他信息，例如在查询中插入了多少行和字节。#5116 (纪尧姆*塔瑟里)

改进

- 已添加 `max_parts_in_total` 设置表的MergeTree家族(默认:100 000)防止分区键的不安全规范#5166. #5171 (阿列克谢-米洛维多夫)
- clickhouse-obfuscator:通过将初始种子与列名（而不是列位置）组合来派生单个列的种子。这用于转换具有多个相关表的数据集，以便在转换后表将保持可联接。#5178 (阿列克谢-米洛维多夫)
- 新增功能 `JSONExtractRaw`, `JSONExtractKeyAndValues`. 重命名函数 `jsonExtract<type>` 到 `JSONExtract<type>`. 当出现问题时，这些函数返回对应的值，而不是 `NULL`. 修改功能 `JSONExtract`，现在它从最后一个参数中获取返回类型，并且不会注入`nullables`。在AVX2指令不可用的情况下实现了回退到RapidJSON。Simdjson库更新到新版本。#5235 (维塔利*巴拉诺夫)
- 现在 `if` 和 `multilf` 功能不依赖于条件的 `Nullable`，但依靠分支来实现sql兼容性。#5238 (吴健)
- `In` 谓词现在生成 `Null` 结果来自 `Null` 输入像 `Equal` 功能。#5152 (吴健)
- 检查来自Kafka的每个 (`flush_interval/poll_timeout`) 行数的时间限制。这允许更频繁地中断Kafka consumer的读取，并检查顶级流的时间限制 #5249 (伊万)
- 链接rdkafka捆绑的SASL。它应该允许使用SASL SCRAM身份验证 #5253 (伊万)
- 所有联接的RowRefList的批处理版本。#5267 (Artem Zuikov)
- clickhouse服务器：更多信息侦听错误消息。#5268 (proller)
- 在clickhouse-复印机的功能支持字典 `<sharding_key>` #5270 (proller)
- 添加新设置 `kafka_commit_every_batch` 来规范卡夫卡的承诺政策。
它允许设置提交模式：在处理每批消息之后，或者在整个块写入存储之后。这是在某些极端情况下丢失一些消息或阅读两次之间的权衡。#5308 (伊万)
- 赖眉露>> `windowFunnel` 支持其他无符号整数类型。#5320 (sundyli)
- 允许对虚拟列进行阴影 `_table` 在合并引擎。#5325 (伊万)
- 赖眉露>> `sequenceMatch` 聚合函数支持其他无符号整数类型 #5339 (sundyli)
- 如果校验和不匹配很可能是由硬件故障引起的，则更好的错误消息。#5355 (阿列克谢-米洛维多夫)
- 检查基础表是否支持以下内容的采样 `StorageMerge` #5366 (伊万)
- Close MySQL connections after their usage in external dictionaries. It is related to issue #893. #5395 (Clément Rodriguez)
- MySQL线协议的改进。将格式名称更改为MySQLWire。使用RAII调用`RSA_free`。如果无法创建上下文，则禁用SSL。#5419 (尤里*巴拉诺夫)
- clickhouse-client: allow to run with unaccessible history file (read-only, no disk space, file is directory, ...). #5431 (proller)
- 尊重异步插入到分布式表中的查询设置。#4936 (TCeason)
- 重命名函数 `leastSqr` 到 `simpleLinearRegression`, `LinearRegression` 到 `linearRegression`, `LogisticRegression` 到 `logisticRegression`. #5391 (尼古拉*科切托夫)

性能改进

- 在ALTER MODIFY查询中并行处理非复制MergeTree表的部分。#4639 (伊万库什)
- Regular表达式提取中的优化。#5193 #5191 (Danila Kutenin)
- 如果仅在join on部分中使用，则不要将右连接键列添加到join result。#5260 (Artem Zuikov)
- 在第一个空响应之后冻结Kafka缓冲区。它避免了多次调用 `ReadBuffer::next()` 对于一些行解析流的空结果。#5283 (伊万)
- `concat` 多个参数的函数优化。#5357 (Danila Kutenin)
- Query optimisation. Allow push down IN statement while rewriting comma/cross join into inner one. #5396 (Artem Zuikov)
- 使用reference one升级我们的LZ4实现以获得更快的解压缩。#5070 (Danila Kutenin)

- 实现了MSD基数排序（基于kxsort）和部分排序。 #5129 (Evgenii Pravda)

错误修复

- 修复推送需要列与联接 #5192 (张冬)
- 修正了当ClickHouse由systemd运行时，命令 `sudo service clickhouse-server forcerestart` 没有按预期工作。 #5204 (proller)
- 修复DataPartsExchange中的http错误代码（9009端口上的服务器间http服务器始终返回代码200，即使是错误）。 #5216 (proller)
- 修复SimpleAggregateFunction字符串长于MAX_SMALL_STRING_SIZE #5311 (Azat Khuzhin)
- 修复错误 Decimal 到 Nullable(Decimal) 转换中。支持其他十进制到十进制转换（包括不同的比例）。 #5350 (Artem Zuikov)
- 修正了simdjson库中导致错误计算的FPU clobbering `uniqHLL` 和 `uniqCombined` 聚合函数和数学函数，如 `log`。 #5354 (阿列克谢-米洛维多夫)
- 固定处理JSON函数中的混合常量/非常量情况。 #5435 (维塔利*巴拉诺夫)
- 修复 `retention` 功能。现在所有满足一行数据的条件都被添加到数据状态。 #5119 (小路)
- 修复结果类型 `quantileExact` 用小数。 #5304 (Artem Zuikov)

文件

- 翻译文档 CollapsingMergeTree 到中国。 #5168 (张风啸)
- 将一些关于表格引擎的文档翻译成中文。
#5134
#5328
(永远不会李)

构建/测试/打包改进

- 修复一些显示可能使用后免费的消毒剂报告。 #5139 #5143 #5393 (伊万)
- 为了方便起见，将性能测试从单独的目录中移出。 #5158 (阿列克谢-米洛维多夫)
- 修复不正确的性能测试。 #5255 (阿利沙平)
- 增加了一个工具来计算由位翻转引起的校验和，以调试硬件问题。 #5334 (阿列克谢-米洛维多夫)
- 使亚军脚本更有用。 #5340#5360 (filimonov)
- 添加如何编写性能测试的小指令。 #5408 (阿利沙平)
- 添加在性能测试中创建，填写和删除查询中进行替换的功能 #5367 (Olga Khvostikova)

ClickHouse释放19.7

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 使用JOIN修复某些查询中的性能回归。 #5192 (张冬)

碌莽禄,拢,010-68520682\<url>戮漏鹿芦,酶,虏卤赂拢,110102003042

新功能

- 添加位图相关功能 `bitmapHasAny` 和 `bitmapHasAll` 类似于 `hasAny` 和 `hasAll` 数组的函数。 #5279 (塞尔吉*弗拉季金)

错误修复

- 修复段错误 `minmax` 具有空值的索引。 #5246 (尼基塔*瓦西列夫)
- 根据需要输出标记"LIMIT BY"中的所有输入列。它修复‘Not found column’某些分布式查询中出错。 #5407 (康斯坦丁*潘)
- 修复“Column ‘0’ already exists”错误 `SELECT .. PREWHERE` 在具有默认值的列上 #5397 (proller)
- 修复 `ALTER MODIFY TTL` 查询开 `ReplicatedMergeTree`。 #5539 (安东*波波夫)
- 当Kafka消费者无法启动时，不要使服务器崩溃。 #5285 (伊万)
- 固定位图函数产生错误的结果。 #5359 (杨小姐)
- 修复散列字典的`element_count`（不包括重复项） #5440 (Azat Khuzhin)
- 使用环境变量TZ的内容作为时区的名称。在某些情况下，它有助于正确检测默认时区。 #5443 (伊万)

- 不要试图将整数转换为 `dictGetT` 功能，因为它不能正常工作。而是抛出一个异常。#5446 (Artem Zuikov)
- 在 `ExternalData` HTTP 请求修复设置。#5455 (Danila
库特宁)
- 修复只从 `FS` 中删除部件而不从 `Zookeeper` 中删除部件时的错误。#5520 (阿利沙平)
- 修复分段故障 `bitmapHasAny` 功能。#5528 (余志昌)
- 修复了复制连接池不重试解析主机时的错误，即使删除了 DNS 缓存。#5534 (阿利沙平)
- 固定 `DROP INDEX IF EXISTS` 查询。现在 `ALTER TABLE ... DROP INDEX IF EXISTS ...` 如果提供的索引不存在，查询不会引发异常。#5524 (格列布·诺维科夫)
- 修复联合所有超类型列。有些情况下，结果列的数据和列类型不一致。#5503 (Artem Zuikov)
- 在 DDL 查询处理过程中跳过 `ZNONODE`。之前，如果另一个节点删除 `znode` 在任务队列中，那一个没有处理它，但已经得到子列表，将终止 `DDLWorker` 线程。#5489 (Azat Khuzhin)
- 修复插入到具体化列的分布式 () 表中。#5429 (Azat Khuzhin)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url> 新功能

- 允许限制用户可以指定的设置的范围。
这些约束可以在用户设置配置文件中设置。
#4931 (维塔利
巴拉诺夫)
- 添加该函数的第二个版本 `groupUniqArray` 用一个可选的 `max_size` 限制结果数组大小的参数。这行为类似于 `groupArray(max_size)(x)` 功能。
#5026 (纪尧姆
Tassery)
- 对于 `TSVWithNames/CSVWithNames` 输入文件格式，列顺序现在可以从文件头确定。这是由控制 `input_format_with_names_use_header` 参数。
#5081
(亚历山大)

错误修复

- 在合并过程中 `uncompressed_cache+JOIN` 崩溃 (#5197)
#5133 (Danila
库特宁)
- `Clickhouse` 客户端查询到系统表上的分段错误。#5066
#5127
(伊万)
- 通过 `KafkaEngine` 重负载数据丢失 (#4736)
#5080
(伊万)
- 修复了在执行 `UNION` 查询时可能发生的非常罕见的数据争用条件，所有查询都涉及至少两个来自系统的选择。列，系统。表，系统。部件，系统。`parts_tables` 或 `Merge` 系列的表，并同时执行相关表的列的更改。#5189 (阿列克谢-米洛维多夫)

性能改进

- 使用基数排序按单个数字列进行排序 `ORDER BY` 没有 `LIMIT`。
#5106,
#4439
(Evgenii Pravda,
阿列克谢-米洛维多夫)

文件

- 将某些表格引擎的文档翻译为中文。
#5107.

#5094,
#5087
(张风啸),
#5068 (从来没有
李)

构建/测试/打包改进

- 正确打印UTF-8字符 clickhouse-test.
#5084
(阿列克谢-米洛维多夫)
- 为 clickhouse-client 添加命令行参数以始终加载建议
戴达 **#5102**
(阿列克谢-米洛维多夫)
- 解决一些PVS-Studio警告。
#5082
(阿列克谢-米洛维多夫)
- 更新LZ4 **#5040** (Danila
库特宁)
- 添加gperf以构建即将到来的拉取请求#5030的requirements。
#5110
(proller)

ClickHouse释放19.6

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复了来自表函数的查询的条件下推 mysql 和 odbc 和相应的表引擎。这修复了#3540和#2384。**#5313** (阿列克谢-米洛维多夫)
- 修复动物园管理员的死锁。**#5297** (github1youlc)
- 允许在CSV中引用小数。**#5284** (Artem Zuikov)
- 禁止从float Inf/NaN转换为小数(抛出异常)。**#5282** (Artem Zuikov)
- 修复重命名查询中的数据竞赛。**#5247** (张冬)
- 暂时禁用LFAlloc。使用LFAlloc可能会导致大量MAP_FAILED在分配UncompressedCache时，并导致高负载服务器上的查询崩溃。**cfdba93**(Danila Kutenin)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能

- 列和表的TTL表达式。**#4212** (安东*波波夫)
- 增加了对 brotli http响应的压缩(接受编码:br) **#4388** (米哈伊尔)
- 增加了新功能 isValidUTF8 用于检查一组字节是否被正确地utf-8编码。**#4934** (Danila Kutenin)
- 添加新的负载平衡策略 first_or_random 它将查询发送到第一个指定的主机,如果无法访问,则向分片的随机主机发送查询。对于跨复制拓扑设置非常有用。**#5012** (纳瓦托洛梅)

实验特点

- 添加设置 index_granularity_bytes (自适应索引粒度)对于MergeTree*表族。**#4826** (阿利沙平)

改进

- 增加了对函数的非常量和负大小和长度参数的支持 substringUTF8。**#4989** (阿列克谢-米洛维多夫)
- 在左联接中禁用向下推到右表,在右联接中禁用左表,并在完全联接中禁用两个表。在某些情况下,这可以修复错误的连接结果。**#4846** (伊万)
- clickhouse-copier:从自动上传任务配置 --task-file 备选案文 **#4876** (proller)
- 为存储工厂和表函数工厂添加了错别字处理程序。**#4891** (Danila Kutenin)
- 支持不带子查询的多个联接的星号和限定星号 **#4898** (Artem Zuikov)
- 伟姑小列链涅消自重加用户友好。**#4915** (Artem Zuikov)

性能改进

- ASOF加速显著 [#4924 \(Martijn Bakker\)](#)

向后不兼容的更改

- HTTP头 `Query-Id` 改名为 `X-ClickHouse-Query-Id` 为了一致性。 [#4972 \(米哈伊尔\)](#)

错误修复

- 修正了潜在的空指针取消引用 `clickhouse-copier`. [#4900 \(proller\)](#)
- 修复了使用`JOIN+ARRAY JOIN`查询的错误 [#4938 \(Artem Zuikov\)](#)
- 固定挂在服务器的启动时，字典依赖于另一个字典通过引擎数据库=字典。 [#4962 \(维塔利·巴拉诺夫\)](#)
- Partially fix `distributed_product_mode = local`. It's possible to allow columns of local tables in `where/having/order by/...` via table aliases. Throw exception if table does not have alias. There's not possible to access to the columns without table aliases yet. [#4986 \(Artem Zuikov\)](#)
- 修复潜在的错误结果 `SELECT DISTINCT` 与 `JOIN` [#5001 \(Artem Zuikov\)](#)
- 修复了在执行`UNION`查询时可能发生的非常罕见的数据争用条件，所有查询都涉及至少两个来自系统的选择。列，系统。表，系统。部件，系统。`parts_tables`或Merge系列的表，并同时执行相关表的列的更改。 [#5189 \(阿列克谢·米洛维多夫\)](#)

构建/测试/打包改进

- 在不同的主机上运行clickhouse服务器时修复测试失败 [#4713 \(瓦西里·内姆科夫\)](#)
- `clickhouse-test`: 在非tty环境中禁用颜色控制序列。 [#4937 \(阿利沙平\)](#)
- `clickhouse-test`: 允许使用任何测试数据库（删除 `test`, 在可能的情况下获得资格）[#5008 \(proller\)](#)
- 修复ubsan错误 [#5037 \(维塔利·巴拉诺夫\)](#)
- Yandex LFAalloc被添加到ClickHouse中，以不同的方式分配MarkCache和UncompressedCache数据，以更可靠地捕获段错误 [#4995 \(Danila Kutenin\)](#)
- Python util帮助反向移植和更改日志。 [#4949 \(伊万\)](#)

ClickHouse释放19.5

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复

- 修正了位图*功能中可能出现的崩溃 [#5220 #5228 \(杨小姐\)](#)
- 修复了在执行`UNION`查询时可能发生的非常罕见的数据争用条件，所有查询都涉及至少两个来自系统的选择。列，系统。表，系统。部件，系统。`parts_tables`或Merge系列的表，并同时执行相关表的列的更改。 [#5189 \(阿列克谢·米洛维多夫\)](#)
- 修正错误 `Set for IN is not created yet in case of using single LowCardinality column in the left part of IN` 如果 `lowcardinality`列是主键的一部分，则会发生此错误。 [#5031 #5154 \(尼古拉·科切托夫\)](#)
- 修改保留函数：如果一行同时满足第一个和第N个条件，则只有第一个满足的条件被添加到数据状态。现在所有满足一行数据的条件都被添加到数据状态。 [#5119 \(小路\)](#)

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复

- 固定设置类型 `max_partitions_per_insert_block` 从布尔到UInt64。 [#5028 \(2.Mohammad Hossein Sekhavat\)](#)

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能

- `超扫描` 添加了多个正则表达式匹配（函数 `multiMatchAny`, `multiMatchAnyIndex`, `multiFuzzyMatchAny`, `multiFuzzyMatchAnyIndex`）。 [#4780, #4841 \(Danila Kutenin\)](#)
- `multiSearchFirstPosition` 添加了功能。 [#4780 \(Danila Kutenin\)](#)
- 为表实现每行的预定义表达式筛选器。 [#4792 \(伊万\)](#)
- 一种基于bloom过滤器的新型数据跳过索引（可用于 `equal`, `in` 和 `like` 功能）。 [#4499 \(尼基塔·瓦西列夫\)](#)
- 已添加 `ASOF JOIN` 它允许运行连接到最新已知值的查询。 [#4774 #4867 #4863 #4875 \(Martijn Bakker, Artem](#)

Zuikov)

- 重写多个 `COMMA JOIN` 到 `CROSS JOIN`. 然后将它们重写为 `INNER JOIN` 如果可能的话 #4661 (Artem Zuikov)

改进

- `topK` 和 `topKWeighted` 现在支持自定义 `loadFactor` (修复问题 #4252). #4634 (基里尔丹信)
- 允许使用 `parallel_replicas_count > 1` 即使对于没有采样的表 (设置简单地忽略它们)。在以前的版本中，它导致异常。#4637 (Alexey Elymanov)
- 支持 `CREATE OR REPLACE VIEW`. 允许在单个语句中创建视图或设置新定义。#4654 (Boris Granveaud)
- `Buffer` 表引擎现在支持 `PREWHERE`. #4671 (刘杨宽)
- 添加在 `zookeeper` 中启动没有元数据的复制表的能力 `readonly` 模式 #4691 (阿利沙平)
- 在 `clickhouse` 客户端固定进度条闪烁。使用时，这个问题最明显 `FORMAT Null` 随着流查询。#4811 (阿列克谢-米洛维多夫)
- 允许禁用功能 `hyperscan` 基于每个用户的库，以限制潜在的过度和不受控制的资源使用。#4816 (阿列克谢-米洛维多夫)
- 添加版本号记录所有错误。#4824 (proller)
- 增加了限制 `multiMatch` 需要字符串大小以适应的函数 `unsigned int`. 还增加了参数的数量限制 `multiSearch` 功能。#4834 (Danila Kutenin)
- 改进了超扫描暂存空间的使用和错误处理。#4866 (Danila Kutenin)
- 填充 `system.graphite_detentions` 从表配置 *GraphiteMergeTree 发动机表. #4584 (Mikhail f. Shiryaev)
- 重命名 `trigramDistance` 功能 `ngramDistance` 并添加更多的功能 `CaseInsensitive` 和 `UTF`. #4602 (Danila Kutenin)
- 改进的数据跳过指数计算。#4640 (尼基塔*瓦西列夫)
- 保持平凡, `DEFAULT`, `MATERIALIZED` 和 `ALIAS` 在一个列表中的列 (修复问题 #2867). #4707 (Alex Zatelepin)

错误修复

- 避免 `std::terminate` 在内存分配失败的情况下。现在 `std::bad_alloc` 按预期引发异常。#4665 (阿列克谢-米洛维多夫)
- 修复 `capnproto` 从缓冲区读取。有时文件没有通过 HTTP 成功加载。#4674 (弗拉季斯拉夫)
- 修复错误 `Unknown log entry type: 0` 后 `OPTIMIZE TABLE FINAL` 查询。#4683 (阿莫斯鸟)
- 错误的参数 `hasAny` 或 `hasAll` 函数可能会导致段错误。#4698 (阿列克谢-米洛维多夫)
- 执行时可能会发生死锁 `DROP DATABASE dictionary` 查询。#4701 (阿列克谢-米洛维多夫)
- 修复未定义的行为 `median` 和 `quantile` 功能。#4702 (hcz)
- 修复压缩级别检测时 `network_compression_method` 小写。在 19.1 节中被打破。#4706 (proller)
- 固定的无知 `<timezone>UTC</timezone>` 设置 (修复问题 #4658). #4718 (proller)
- 修复 `histogram` 函数行为 `Distributed` 桌子 #4741 (olegkv)
- 固定 `tsan` 报告 `destroy of a locked mutex`. #4742 (阿列克谢-米洛维多夫)
- 修复了由于系统日志使用中的争用条件而关闭的 `TSan` 报告。修复了当 `part_log` 启用时关机后的潜在使用。#4758 (阿列克谢-米洛维多夫)
- 修复重新检查零件 `ReplicatedMergeTreeAlterThread` 在错误的情况下。#4772 (尼古拉*科切托夫)
- 对中间聚合函数状态的算术运算不适用于常量参数 (如子查询结果)。#4776 (阿列克谢-米洛维多夫)
- 始终在元数据中反引用列名。否则，不可能创建一个名为列的表 `index` (由于格式错误，服务器无法重新启动 `ATTACH` 元数据中的查询)。#4782 (阿列克谢-米洛维多夫)
- 修复崩溃 `ALTER ... MODIFY ORDER BY` 上 `Distributed` 桌子 #4790 (TCeason)
- 修复段错误 `JOIN ON` 已启用 `enable_optimize_predicate_expression`. #4794 (张冬)
- 修复 `kafka` 使用 `protobuf` 消息后添加无关行的错误。#4808 (维塔利*巴拉诺夫)
- 修复崩溃 `JOIN` 在不可为空的 `vs` 可为空的列上。修复 `NULLs` 在右键 `ANY JOIN + join_use_nulls`. #4815 (Artem Zuikov)
- 修复分段故障 `clickhouse-copier`. #4835 (proller)
- 在固定的竞争条件 `SELECT` 从 `system.tables` 如果同时重命名或更改表。#4836 (阿列克谢-米洛维多夫)
- 获取已经过时的数据部分时修复了数据竞赛。#4839 (阿列克谢-米洛维多夫)
- 固定罕见的数据竞赛，可以在发生 `RENAME` `MergeTree` 家族的表。#4844 (阿列克谢-米洛维多夫)
- 修正功能中的分段故障 `arrayIntersect`. 如果函数使用常量和普通参数混合调用，则可能会发生分段错误。#4847 (钱丽祥)
- 固定读取 `Array(LowCardinality)` column 在极少数情况下，当 `column` 包含一个长序列的空数组时。#4850 (尼古拉*科切托夫)
- 修复崩溃 `FULL/RIGHT JOIN` 当我们加入可为空 `vs` 不可为空时。#4855 (Artem Zuikov)

- 修复 No message received 在副本之间获取部件时出现异常。 #4856 (阿利沙平)
- 固定 arrayIntersect 函数错误导致在单个数组中的几个重复值的情况下。 #4871 (尼古拉*科切托夫)
- 在并发期间修复争用条件 ALTER COLUMN 可能导致服务器崩溃的查询 (修复问题 #3421). #4592 (Alex Zatelepin)
- 修复不正确的结果 FULL/RIGHT JOIN 与常量列。 #4723 (Artem Zuikov)
- 修复重复 GLOBAL JOIN 用星号。 #4705 (Artem Zuikov)
- 修复参数扣除 ALTER MODIFY 列 CODEC 未指定列类型时。 #4883 (阿利沙平)
- 功能 cutQueryStringAndFragment() 和 queryStringAndFragment() 现在正常工作时 URL 包含一个片段，没有查询。 #4894 (维塔利*巴拉诺夫)
- 修复设置时罕见的错误 min_bytes_to_use_direct_io 大于零，这发生在线程必须在列文件中向后寻找时。 #4897 (阿利沙平)
- 修复聚合函数的错误参数类型 LowCardinality 参数 (修复问题 #4919). #4922 (尼古拉*科切托夫)
- 修复错误的名称资格 GLOBAL JOIN. #4969 (Artem Zuikov)
- 修复功能 toISOWeek 1970年的结果。 #4988 (阿列克谢-米洛维多夫)
- 修复 DROP, TRUNCATE 和 OPTIMIZE 查询重复，在执行时 ON CLUSTER 为 ReplicatedMergeTree* 表家庭. #4991 (阿利沙平)

向后不兼容的更改

- 重命名设置 insert_sample_with_metadata 到设置 input_format_defaults_for_omitted_fields. #4771 (Artem Zuikov)
- 添加设置 max_partitions_per_insert_block (默认值为 100)。如果插入的块包含较大量数的分区，则会引发异常。如果要删除限制 (不推荐)，请将其设置为 0。 #4845 (阿列克谢-米洛维多夫)
- 多搜索功能被重命名 (multiPosition 到 multiSearchAllPositions, multiSearch 到 multiSearchAny, firstMatch 到 multiSearchFirstIndex). #4780 (Danila Kutenin)

性能改进

- 通过内联优化Volnitsky搜索器，为许多针或许多类似bigrams的查询提供约5-10%的搜索改进。 #4862 (Danila Kutenin)
- 修复设置时的性能问题 use_uncompressed_cache 大于零时，即出现在所有读取缓存中包含的数据时。 #4913 (阿利沙平)

构建/测试/包装改进

- 强化调试构建：更精细的内存映射和ASLR;为标记缓存和索引添加内存保护。这允许在ASan和MSan无法做到这一点的情况下找到更多的内存st脚错误。 #4632 (阿列克谢-米洛维多夫)
- 添加对cmake变量的支持 ENABLE_PROTOBUF, ENABLE_PARQUET 和 ENABLE_BROTLI 它允许启用/禁用上述功能 (与我们对librdkafka, mysql等所做的相同)。 #4669 (Silviu Caragea)
- 添加打印进程列表和堆栈跟踪的所有线程的能力，如果一些查询测试运行后挂起。 #4675 (阿利沙平)
- 添加重试 Connection loss 错误 clickhouse-test. #4682 (阿利沙平)
- 在打包程序脚本中添加使用 vagrant 的 freebsd build 和使用 thread sanitizer 的 build。 #4712 #4748 (阿利沙平)
- 现在用户要求用户密码 'default' 在安装过程中。 #4725 (proller)
- 禁止在警告 rdkafka 图书馆. #4740 (阿列克谢-米洛维多夫)
- 允许在没有ssl的情况下构建。 #4750 (proller)
- 添加从自定义用户启动clickhouse服务器映像的方法。 #4753 (Mikhail f. Shiryaev)
- 升级contrib升压到1.69. #4793 (proller)
- 禁用使用 mremap 使用线程消毒剂编译时。令人惊讶的是，TSan并没有拦截 mremap (虽然它确实拦截 mmap, munmap 这会导致误报。修复了有状态测试中的TSan报告。 #4859 (阿列克谢-米洛维多夫)
- 通过HTTP接口使用格式模式添加测试检查。 #4864 (维塔利*巴拉诺夫)

ClickHouse释放19.4

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 避免 std::terminate 在内存分配失败的情况下。现在 std::bad_alloc 按预期引发异常。 #4665 (阿列克谢-米洛维多夫)
- 修复capnproto从缓冲区读取。有时文件没有通过HTTP成功加载。 #4674 (弗拉季斯拉夫)
- 修复错误 Unknown log entry type: 0 后 OPTIMIZE TABLE FINAL 查询。 #4683 (阿莫斯鸟)
- 错误的参数 hasAny 或 hasAll 函数可能会导致段错误。 #4698 (阿列克谢-米洛维多夫)
- 执行时可能会发生死锁 DROP DATABASE dictionary 查询。 #4701 (阿列克谢-米洛维多夫)

- 修复未定义的行为 median 和 quantile 功能。 #4702 (hc2)
- 修复压缩级别检测时 network_compression_method 小写。在19.1节中被打破。 #4706 (proller)
- 固定的无知 <timezone>UTC</timezone> 设置 (修复问题 #4658). #4718 (proller)
- 修复 histogram 函数行为 Distributed 桌子 #4741 (olegkv)
- 固定tsan报告 destroy of a locked mutex. #4742 (阿列克谢-米洛维多夫)
- 修复了由于系统日志使用中的争用条件而关闭的TSan报告。修复了当part_log启用时关机后的潜在使用。 #4758 (阿列克谢-米洛维多夫)
- 修复重新检查零件 ReplicatedMergeTreeAlterThread 在错误的情况下。 #4772 (尼古拉*科切托夫)
- 对中间聚合函数状态的算术运算不适用于常量参数 (如子查询结果)。 #4776 (阿列克谢-米洛维多夫)
- 始终在元数据中反引用列名。否则，不可能创建一个名为列的表 index (由于格式错误，服务器无法重新启动 ATTACH 元数据中的查询)。 #4782 (阿列克谢-米洛维多夫)
- 修复崩溃 ALTER ... MODIFY ORDER BY 上 Distributed 桌子 #4790 (TCeason)
- 修复段错误 JOIN ON 已启用 enable_optimize_predicate_expression. #4794 (张冬)
- 修复kafka使用protobuf消息后添加无关行的错误。 #4808 (维塔利*巴拉诺夫)
- 修复分段故障 clickhouse-copier. #4835 (proller)
- 在固定的竞争条件 SELECT 从 system.tables 如果同时重命名或更改表。 #4836 (阿列克谢-米洛维多夫)
- 获取已经过时的数据部分时修复了数据竞赛。 #4839 (阿列克谢-米洛维多夫)
- 固定罕见的数据竞赛，可以在发生 RENAME MergeTree家族的表. #4844 (阿列克谢-米洛维多夫)
- 修正功能中的分段故障 arrayIntersect. 如果函数使用常量和普通参数混合调用，则可能会发生分段错误。 #4847 (钱丽祥)
- 固定读取 Array(LowCardinality) column在极少数情况下，当column包含一个长序列的空数组时。 #4850 (尼古拉*科切托夫)
- 修复 No message received 在副本之间获取部件时出现异常。 #4856 (阿利沙平)
- 固定 arrayIntersect 函数错误导致在单个数组中的几个重复值的情况下。 #4871 (尼古拉*科切托夫)
- 在并发期间修复争用条件 ALTER COLUMN 可能导致服务器崩溃的查询 (修复问题 #3421). #4592 (Alex Zatelepin)
- 修复参数扣除 ALTER MODIFY 列 CODEC 未指定列类型时。 #4883 (阿利沙平)
- 功能 cutQueryStringAndFragment() 和 queryStringAndFragment() 现在正常工作时 URL 包含一个片段，没有查询。 #4894 (维塔利*巴拉诺夫)
- 修复设置时罕见的错误 min_bytes_to_use_direct_io 大于零，这发生在线程必须在列文件中向后寻找时。 #4897 (阿利沙平)
- 修复聚合函数的错误参数类型 LowCardinality 参数 (修复问题 #4919). #4922 (尼古拉*科切托夫)
- 修复功能 toISOWeek 1970年的结果。 #4988 (阿列克谢-米洛维多夫)
- 修复 DROP, TRUNCATE 和 OPTIMIZE 查询重复，在执行时 ON CLUSTER 为 ReplicatedMergeTree* 表家庭. #4991 (阿利沙平)

改进

- 保持平凡，DEFAULT, MATERIALIZED 和 ALIAS 在一个列表中的列 (修复问题 #2867). #4707 (Alex Zatelepin)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复崩溃 FULL/RIGHT JOIN 当我们加入可为空vs不可为空时。 #4855 (Artem Zuikov)
- 修复分段故障 clickhouse-copier. #4835 (proller)

构建/测试/包装改进

- 添加从自定义用户启动clickhouse服务器映像的方法。 #4753 (Mikhail f. Shiryaev)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 固定读取 Array(LowCardinality) column在极少数情况下，当column包含一个长序列的空数组时。 #4850 (尼古拉*科切托夫)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 包含两个固定的远程查询 `LIMIT BY` 和 `LIMIT`. 以前, 如果 `LIMIT BY` 和 `LIMIT` 用于远程查询, `LIMIT` 可能发生之前 `LIMIT BY`, 这导致过滤的结果。#4708 (康斯坦丁*潘)

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能

- 增加了全面支持 `Protobuf` 格式 (输入和输出, 嵌套数据结构) 。#4174 #4493 (维塔利*巴拉诺夫)
- 添加位图功能与Ro嗦的位图。#4207 (杨小姐) #4568 (维塔利*巴拉诺夫)
- 实木复合地板格式支持。#4448 (proller)
- 为模糊字符串比较添加了N-gram距离。它类似于R语言中的q-gram指标。#4466 (Danila Kutenin)
- 结合专用聚合和保留模式中的石墨汇总规则。#4426 (Mikhail f. Shiryaev)
- 已添加 `max_execution_speed` 和 `max_execution_speed_bytes` 限制资源使用。已添加 `min_execution_speed_bytes` 设置以补充 `min_execution_speed`. #4430 (张冬)
- 实现功能 `flatten`. #4555 #4409 (阿列克谢-米洛维多夫, kzon)
- 新增功能 `arrayEnumerateDenseRanked` 和 `arrayEnumerateUniqRanked` (这就像 `arrayEnumerateUniq` 但是允许微调数组深度以查看多维数组内部) 。#4475 (proller) #4601 (阿列克谢-米洛维多夫)
- Multiple JOINS with some restrictions: no asterisks, no complex aliases in ON/WHERE/GROUP BY/... #4462 (Artem Zuikov)

错误修复

- 此版本还包含19.3和19.1中的所有错误修复。
- 修正了数据跳过索引的错误：插入后颗粒顺序不正确。#4407 (尼基塔*瓦西列夫)
- 固定 `set` 环板`for` `Nullable` 和 `LowCardinality` 列。在它之前, `set` 索引与 `Nullable` 或 `LowCardinality` 列导致错误 `Data type must be deserialized with multiple streams` 同时选择。#4594 (尼古拉*科切托夫)
- 正确设置完整的`update_time executable` 字典更新. #4551 (Tema Novikov)
- 修复19.3中损坏的进度条。#4627 (filimonov)
- 在某些情况下，修复了内存区域收缩时MemoryTracker的不一致值。#4619 (阿列克谢-米洛维多夫)
- 修复了ThreadPool中未定义的行为。#4612 (阿列克谢-米洛维多夫)
- 修正了一个非常罕见的崩溃的消息 `mutex lock failed: Invalid argument` 当 MergeTree表与SELECT同时删除时，可能会发生这种情况。#4608 (Alex Zatelepin)
- ODBC驱动程序兼容 `LowCardinality` 数据类型。#4381 (proller)
- FreeBSD:修复程序 `AIOContextPool: Found io_event with unknown id 0` 错误 #4438 (urgordeadbeef)
- `system.part_log` 无论配置如何，都会创建表。#4483 (阿列克谢-米洛维多夫)
- 修复未定义的行为 `dictIsIn` 缓存字典功能。#4515 (阿利沙平)
- Fixed a deadlock when a SELECT query locks the same table multiple times (e.g. from different threads or when executing multiple subqueries) and there is a concurrent DDL query. #4535 (Alex Zatelepin)
- 默认情况下禁用`compile_expressions`，直到我们得到自己 `llvm contrib`并且可以测试它 `clang` 和 `asan`. #4579 (阿利沙平)
- 预防 `std::terminate` 当 `invalidate_query` 为 `clickhouse` 外部字典源返回了错误的结果集（空或一行以上或一列以上）。固定的问题，当 `invalidate_query` 执行每五秒钟，无论到 `lifetime`. #4583 (阿列克谢-米洛维多夫)
- 避免死锁时 `invalidate_query` 对于与字典 `clickhouse` 资料来源涉及 `system.dictionaries` 表或 `Dictionaries` 数据库（罕见的情况）。#4599 (阿列克谢-米洛维多夫)
- 修复了交叉连接与空在哪里。#4598 (Artem Zuikov)
- 在功能固定段错误“`replicate`”传递常量参数时。#4603 (阿列克谢-米洛维多夫)
- 使用谓词优化器修复`lambda`函数。#4408 (张冬)
- 多个联接多个修复。#4595 (Artem Zuikov)

改进

- 在右表列的连接上部分支持别名。#4412 (Artem Zuikov)
- 多个联接的结果需要在子选择中使用正确的结果名称。将平面别名替换为`result`中的源名称。#4474 (Artem Zuikov)
- 改进连接语句的下推逻辑。#4387 (伊万)

性能改进

- 改进的启发式“`move to PREFWHREF`”优化。#4405 (阿列克谢-米洛维多夫)

- 使用适当的查找表，使用HashTable的api用于8位和16位密钥。 #4536 (阿莫斯鸟)
- 改进字符串比较的性能。 #4564 (阿列克谢-米洛维多夫)
- 在单独的线程中清理分布式DDL队列，以便它不会减慢处理分布式DDL任务的主循环。 #4502 (Alex Zatelepin)
- 当 `min_bytes_to_use_direct_io` 如果设置为1，则不是每个文件都使用O_DIRECT模式打开，因为要读取的数据大小有时被一个压缩块的大小所低估。 #4526 (阿列克谢-米洛维多夫)

构建/测试/包装改进

- 增加了对clang-9的支持 #4604 (阿列克谢-米洛维多夫)
- 修复错误 `_asm_` 说明 (再次) #4621 (Konstantin Podshumok)
- 添加指定设置的能力 `clickhouse-performance-test` 从命令行。 #4437 (阿利沙平)
- 将字典测试添加到集成测试。 #4477 (阿利沙平)
- 在网站上添加了来自基准测试的查询，以自动化性能测试。 #4496 (阿列克谢-米洛维多夫)
- `xxhash.h` 在外部lz4中不存在，因为它是一个实现细节，并且它的符号是命名空间的 XXH_NAMESPACE 麦克罗 当lz4是外部的，xxHash也必须是外部的，并且依赖者必须链接到它。 #4495 (Origej Desh)
- 固定的情况下，当 `quantileTiming` 聚合函数可以用负或浮点参数调用 (这修复了使用未定义的行为消毒器的模糊测试)。 #4506 (阿列克谢-米洛维多夫)
- 拼写错误更正。 #4531 (sdk2)
- 在Mac上修复编译。 #4371 (维塔利*巴拉诺夫)
- Freebsd和各种不寻常的构建配置的构建修复程序。 #4444 (proller)

ClickHouse释放19.3

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 修复崩溃 `FULL/RIGHT JOIN` 当我们加入可为空vs不可为空时。 #4855 (Artem Zuikov)
- 修复分段故障 `clickhouse-copier`. #4835 (proller)
- 固定读取 `Array(LowCardinality)` column在极少数情况下，当column包含一个长序列的空数组时。 #4850 (尼古拉*科切托夫)

构建/测试/包装改进

- 添加从自定义用户启动clickhouse服务器映像的方法 #4753 (Mikhail f. Shiryaev)

碌莽禄,拢,010-68520682\<url>

错误修复

- 修正了#3920中的错误。此错误表现为随机缓存损坏 (消息 `Unknown codec family code, Cannot seek through file`) 和段错误。这个错误最早出现在19.1版本中，并且存在于19.1.10和19.3.6之前的版本中。 #4623 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复

- 当线程池中有超过1000个线程时，`std::terminate` 线程退出时可能发生。 Azat Khuzhin #4485 #4505 (阿列克谢-米洛维多夫)
- 现在可以创建 `ReplicatedMergeTree*` 对没有默认值的列进行注释的表和对没有注释和默认值的列进行编解码的表。还修复编解码器的比较。 #4523 (阿利沙平)
- 修复了与数组或元组联接时的崩溃。 #4552 (Artem Zuikov)
- 修复了clickhouse-复印机中的消息崩溃 `ThreadStatus not created.` #4540 (Artem Zuikov)
- 如果使用分布式Ddl，则在服务器关闭时修复了挂机问题。 #4472 (Alex Zatelepin)
- 错误的列编号打印在有关文本格式分析的列数大于10的错误消息中。 #4484 (阿列克谢-米洛维多夫)

构建/测试/打包改进

- 固定构建与启用AVX。 #4527 (阿列克谢-米洛维多夫)
- 基于已知版本而不是编译它的内核启用扩展记帐和IO记帐。 #4541 (纳瓦托洛梅)
- 允许跳过core_dump的设置。`size_limit`，如果限制设置失败，则警告而不是throw。 #4473 (proller)

- 删除了 `inline` 标签 `void readBinary(...)` 在 `Field.cpp`. 也合并冗余 `namespace DB` 块。 #4530 (hcz)

碌莽禄,拢,010-68520682\<url>

错误修复

- 修正了大型http插入查询处理的错误。 #4454 (阿利沙平)
- 修正了向后不兼容的旧版本，由于错误的实现 `send_logs_level` 设置。 #4445 (阿列克谢-米洛维多夫)
- 修正了表函数的向后不兼容性 `remote` 与列注释介绍. #4446 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682\<url>

改进

- 执行以下操作时，表索引大小不考虑内存限制 `ATTACH TABLE` 查询。避免了分离后无法连接表的可能性。 #4396 (阿列克谢-米洛维多夫)
- 稍微提高了从ZooKeeper接收的最大字符串和数组大小的限制。它允许继续与增加的尺寸工作 `CLIENT_JVMFLAGS=-Djute.maxbuffer=...` 在动物园管理员。 #4398 (阿列克谢-米洛维多夫)
- 允许修复被遗弃的副本，即使它已经在其队列中拥有大量的节点。 #4399 (阿列克谢-米洛维多夫)
- 添加一个必需的参数 `SET` 索引 (最大存储行数) 。 #4386 (尼基塔*瓦西列夫)

错误修复

- 固定 `WITH ROLLUP` 单组结果 `LowCardinality` 钥匙 #4384 (尼古拉*科切托夫)
- 在设置索引固定错误 (删除颗粒，如果它包含超过 `max_rows` 行) 。 #4386 (尼基塔*瓦西列夫)
- 很多的FreeBSD构建修复。 #4397 (proller)
- 固定别名替换查询与子查询包含相同的别名 (问题 #4110). #4351 (Artem Zuikov)

构建/测试/打包改进

- 添加运行能力 `clickhouse-server` 对于docker镜像中的无状态测试。 #4347 (瓦西里*内姆科夫)

碌莽禄,拢,010-68520682\<url>

新功能

- 添加了 `KILL MUTATION` 允许删除由于某些原因卡住的突变的声明。已添加 `latest_failed_part`, `latest_fail_time`, `latest_fail_reason` 字段到 `system.mutations` 表更容易排除故障。 #4287 (Alex Zatelepin)
- 添加聚合功能 `entropy` 计算香农熵 #4238 (Quid37)
- 添加发送查询的功能 `INSERT INTO tbl VALUES (....` 到服务器而不拆分 `query` 和 `data` 零件。 #4301 (阿利沙平)
- 通用实现 `arrayWithConstant` 添加了功能。 #4322 (阿列克谢-米洛维多夫)
- 已实施 `NOT BETWEEN` 比较运算符。 #4228 (Dmitry Naumov)
- 执行 `sumMapFiltered` 为了能够限制其值将被求和的键的数量 `sumMap`. #4129 (Léo Ercolanelli)
- 增加了支持 `Nullable` 类型 `mysql` 表功能。 #4198 (Emmanuel Donin de Rosière)
- 支持任意常量表达式 `LIMIT` 条款 #4246 (k3box)
- 已添加 `topKWeighted` 采用带有 (无符号整数) 权重的附加参数的聚合函数。 #4245 (安德鲁*戈尔曼)
- `StorageJoin` 现在支持 `join_any_take_last_row` 允许复盖同一键的现有值的设置。 #3973 (阿莫斯鸟)
- 添加功能 `toStartOfInterval`. #4304 (维塔利*巴拉诺夫)
- 已添加 `RowBinaryWithNamesAndTypes` 格式。 #4200 (Oleg V.Kozlyuk)
- 已添加 `IPv4` 和 `IPv6` 数据类型。更有效的实现 `IPv*` 功能。 #3669 (瓦西里*内姆科夫)
- 添加功能 `toStartOfTenMinutes()`. #4298 (维塔利*巴拉诺夫)
- 已添加 `Protobuf` 输出格式。 #4005 #4158 (维塔利*巴拉诺夫)
- 增加了对数据导入 (插入) HTTP接口的brotli支持。 #4235 (米哈伊尔)
- 增加了提示，而用户做出错字的函数名称或键入命令行客户端。 #4239 (Danila Kutenin)
- 已添加 `Query-Id` 到服务器的HTTP响应头。 #4231 (米哈伊尔)

实验特点

- 已添加 `minmax` 和 `set` MergeTree表引擎系列的数据跳过索引。 #4143 (尼基塔*瓦西列夫)
- 增加了转换 `CROSS JOIN` 到 `INNER JOIN` 如果可能的话 #4221 #4266 (Artem Zuikov)

错误修复

- 固定 Not found column 对于重复的列 JOIN ON 科。 #4279 (Artem Zuikov)
- 赖眉露>> START REPLICATED SENDS 命令开始复制发送。 #4229 (纳瓦托洛梅)
- 固定聚合函数执行 Array(LowCardinality) 争论。 #4055 (KochetovNicolai)
- 修正了错误的行为，当做 INSERT ... SELECT ... FROM file(...) 查询和文件有 CSVWithNames 或 TSVWithNames 格式和第一个数据行丢失。 #4297 (阿列克谢-米洛维多夫)
- 如果字典不可用，则修复了字典重新加载时的崩溃。此错误出现在19.1.6中。 #4188 (proller)
- 固定 ALL JOIN 右表中有重复项。 #4184 (Artem Zuikov)
- 修正了分段故障 use_uncompressed_cache=1 和异常与错误的未压缩大小。此错误出现在19.1.6中。 #4186 (阿利沙平)
- 固定 compile_expressions 错误与大（超过int16）日期的比较。 #4341 (阿利沙平)
- 从表函数选择时固定无限循环 numbers(0). #4280 (阿列克谢-米洛维多夫)
- 暂时禁用谓词优化 ORDER BY. #3890 (张冬)
- 固定 Illegal instruction 在旧Cpu上使用base64函数时出错。仅当ClickHouse使用gcc-8编译时，才会重现此错误。 #4275 (阿列克谢-米洛维多夫)
- 固定 No message received 通过TLS连接与PostgreSQL ODBC驱动程序交互时出错。还修复了使用MySQL ODBC驱动程序时的段错误。 #4170 (阿列克谢-米洛维多夫)
- 修正错误的结果时 Date 和 DateTime 参数用于条件运算符（函数）的分支 if. 增加了函数的通用案例 if. #4243 (阿列克谢-米洛维多夫)
- ClickHouse字典现在加载内 clickhouse 过程。 #4166 (阿列克谢-米洛维多夫)
- 修复死锁时 SELECT 从一个表 File 引擎被重试后 No such file or directory 错误 #4161 (阿列克谢-米洛维多夫)
- 从选择时固定的竞争条件 system.tables 可能会给 table doesn't exist 错误 #4313 (阿列克谢-米洛维多夫)
- clickhouse-client 如果在交互模式下运行，则在加载命令行建议的数据时可以在退出时段错误。 #4317 (阿列克谢-米洛维多夫)
- 修正了一个错误，当包含突变的执行 IN 操作员产生了不正确的结果。 #4099 (Alex Zatelepin)
- 修正错误：如果有一个数据库 Dictionary 引擎中，所有字典在服务器启动时强制加载，如果有来自localhost的ClickHouse源字典，则字典无法加载。 #4255 (阿列克谢-米洛维多夫)
- 修复了在服务器关闭时尝试再次创建系统日志时的错误。 #4254 (阿列克谢-米洛维多夫)
- 正确返回正确的类型和正确处理锁 joinGet 功能。 #4153 (阿莫斯鸟)
- 已添加 sumMapWithOverflow 功能。 #4151 (Léo Ercolanelli)
- 固定段错误 allow_experimental_multiple_joins_emulation. 52de2c (Artem Zuikov)
- 修正错误与不正确 Date 和 DateTime 比较。 #4237 (valexey)
- 在未定义的行为消毒固定模糊测试：增加了参数类型检查 quantile*Weighted 家庭的功能。 #4145 (阿列克谢-米洛维多夫)
- 修复了在删除旧数据部分时罕见的争用条件可能会失败 File not found 错误 #4378 (阿列克谢-米洛维多夫)
- 修复缺少/etc/clickhouse-server/config的安装包。xml #4343 (proller)

构建/测试/打包改进

- Debian软件包：根据配置正确的/etc/clickhouse-server/预处理链接。 #4205 (proller)
- Freebsd的各种构建修复程序。 #4225 (proller)
- 增加了在perftest中创建，填充和删除表的能力。 #4220 (阿利沙平)
- 添加了一个脚本来检查重复的包括。 #4326 (阿列克谢-米洛维多夫)
- 增加了在性能测试中通过索引运行查询的能力。 #4264 (阿利沙平)
- 建议安装带有调试符号的软件包。 #4274 (阿列克谢-米洛维多夫)
- 重构性能测试。更好的记录和信号处理。 #4171 (阿利沙平)
- 将文档添加到匿名Yandex。Metrika数据集. #4164 (阿利沙平)
- Added tool for converting an old month-partitioned part to the custom-partitioned format. #4195 (Alex Zatelepin)
- 添加了有关s3中两个数据集的文档。 #4144 (阿利沙平)
- 增加了从拉请求描述创建更新日志的脚本。 #4169 #4173 (KochetovNicolai) (KochetovNicolai)
- 为ClickHouse添加了木偶模块。 #4182 (Maxim Fedotov)
- 添加了一组无证函数的文档。 #4168 (张冬)
- ARM构建修复。 #4210 #4306 #4291 (proller) (proller)
- 字典测试现在能够从运行 ctest. #4189 (proller)
- 现在 /etc/ssl 用作带有SSL证书的默认目录。 #4167 (阿列克谢-米洛维多夫)

- 在开始时添加了检查 `SSCE` 和 `AVX` 指令。 #4234 (igr)
- 初始化脚本将等待服务器，直到启动。 #4281 (proller)

向后不兼容的更改

- 已删除 `allow_experimental_low_cardinality_type` 设置。 `LowCardinality` 数据类型已准备就绪。 #4323 (阿列克谢-米洛维多夫)
- 根据可用内存量减少标记高速缓存大小和未压缩高速缓存大小。 #4240 (Lopatin Konstantin)
- 添加关键字 `INDEX` 在 `CREATE TABLE` 查询。 具有名称的列 `index` 必须使用反引号或双引号引用: `index`。 #4143 (尼基塔*瓦西列夫)
- `sumMap` 现在提升结果类型而不是溢出。 老 `sumMap` 行为可以通过使用获得 `sumMapWithOverflow` 功能。 #4151 (Léo Ercolanelli)

性能改进

- `std::sort` 改为 `pdqsort` 对于没有 `LIMIT`。 #4236 (Evgenii Pravda)
- 现在服务器重用全局线程池中的线程。 这会影响某些角落情况下的性能。 #4150 (阿列克谢-米洛维多夫)

改进

- 实现了对FreeBSD的AIO支持。 #4305 (urgordeadbeef)
- `SELECT * FROM a JOIN b USING a, b` 现在回来 `a` 和 `b` 列仅从左表。 #4141 (Artem Zuikov)
- 允许 `-C` 客户端的选项作为工作 `-c` 选项。 #4232 (syominsergey)
- 现在选项 `--password` 无值使用需要从标准输入的密码。 #4230 (BSD_Conqueror)
- 在包含字符串文字中添加了非转义元字符的突出显示 `LIKE` 表达式或正则表达式。 #4327 (阿列克谢-米洛维多夫)
- 添加取消HTTP只读查询，如果客户端套接字消失。 #4213 (纳瓦托洛梅)
- 现在，服务器报告进度，以保持客户端连接活跃。 #4215 (伊万)
- 稍微好一点的消息与优化查询的原因 `optimize_throw_if_noop` 设置已启用。 #4294 (阿列克谢-米洛维多夫)
- 增加了支持 `--version` `clickhouse` 服务器的选项。 #4251 (Lopatin Konstantin)
- 已添加 `--help/-h` 选项 `clickhouse-server`。 #4233 (尤里*巴拉诺夫)
- 增加了对具有聚合函数状态结果的标量子查询的支持。 #4348 (尼古拉*科切托夫)
- 改进服务器关闭时间并改变等待时间。 #4372 (阿列克谢-米洛维多夫)
- 添加了有关 `replicated_can_become_leader` 设置到系统的信息。如果副本不会尝试成为领导者，则添加日志记录。 #4379 (Alex Zatelepin)

ClickHouse释放19.1

碌莽禄,拢,010-68520682\<url>

- 修正错误 `Column ... queried more than once` 这可能发生在，如果设置 `asterisk_left_columns_only` 在使用的情况下设置为 1 `GLOBAL JOIN` 与 `SELECT *` (罕见的情况)。该问题在19.3及更新版本中不存在。 6bac7d8d (Artem Zuikov)

碌莽禄,拢,010-68520682\<url>

此版本包含与19.3.7完全相同的补丁集。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

此版本包含与19.3.6完全相同的补丁集。

ClickHouse释放19.1

碌莽禄,拢,010-68520682\<url>

错误修复

- 修正了向后不兼容的旧版本，由于错误的实现 `send_logs_level` 设置。 #4445 (阿列克谢-米洛维多夫)
- 修正了表函数的向后不兼容性 `remote` 与列注释介绍。 #4446 (阿列克谢-米洛维多夫)

碌莽禄,拢,010-68520682\<url>

错误修复

- 修复缺少`/etc/clickhouse-server/config`的安装包。xml #4343 (proller)

ClickHouse 释放 19.1

碌莽禄, 拢, 010-68520682\<url>

错误修复

- 正确返回正确的类型和正确处理锁 `joinGet` 功能。 #4153 (阿莫斯鸟)
- 修复了在服务器关闭时尝试再次创建系统日志时的错误。 #4254 (阿列克谢-米洛维多夫)
- 修正错误：如果有一个数据库 `Dictionary` 引擎中，所有字典在服务器启动时强制加载，如果有来自 `localhost` 的 ClickHouse 源字典，则字典无法加载。 #4255 (阿列克谢-米洛维多夫)
- 修正了一个错误，当包含突变的执行 `IN` 操作员产生了不正确的结果。 #4099 (Alex Zatelepin)
- `clickhouse-client` 如果在交互模式下运行，则在加载命令行建议的数据时可以在退出时段错误。 #4317 (阿列克谢-米洛维多夫)
- 从选择时固定的竞争条件 `system.tables` 可能会给 `table doesn't exist` 错误 #4313 (阿列克谢-米洛维多夫)
- 修复死锁时 `SELECT` 从一个表 `File` 引擎被重试后 `No such file or directory` 错误 #4161 (阿列克谢-米洛维多夫)
- 修复了一个问题：本地 ClickHouse 字典通过 TCP 加载，但应该在进程中加载。 #4166 (阿列克谢-米洛维多夫)
- 固定 `No message received` 通过 TLS 连接与 PostgreSQL ODBC 驱动程序交互时出错。还修复了使用 MySQL ODBC 驱动程序时的段错误。 #4170 (阿列克谢-米洛维多夫)
- 暂时禁用谓词优化 `ORDER BY`. #3890 (张冬)
- 从表函数选择时固定无限循环 `numbers(0)`. #4280 (阿列克谢-米洛维多夫)
- 固定 `compile_expressions` 错误与大（超过 `int16`）日期的比较。 #4341 (阿利沙平)
- 修正了分段故障 `uncompressed_cache=1` 和异常与错误的未压缩大小。 #4186 (阿利沙平)
- 固定 `ALL JOIN` 右表中有重复项。 #4184 (Artem Zuikov)
- 修正了错误的行为，当做 `INSERT ... SELECT ... FROM file(...)` 查询和文件有 `CSVWithNames` 或 `TSVWithNames` 格式和第一个数据行丢失。 #4297 (阿列克谢-米洛维多夫)
- 固定聚合函数执行 `Array(LowCardinality)` 争论。 #4055 (KochetovNicolai)
- Debian 软件包：根据配置正确的 /etc/clickhouse-server/ 预处理链接。 #4205 (proller)
- 在未定义的行为消毒固定模糊测试：增加了参数类型检查 `quantile*Weighted` 家庭的功能。 #4145 (阿列克谢-米洛维多夫)
- 眉露>> `START REPLICATED SENDS` 命令开始复制发送。 #4229 (纳瓦托洛梅)
- 固定 `Not found column` 对于联接部分中的重复列。 #4279 (Artem Zuikov)
- 现在 /etc/ssl 用作带有 SSL 证书的默认目录。 #4167 (阿列克谢-米洛维多夫)
- 如果字典不可用，则修复了字典重新加载时的崩溃。 #4188 (proller)
- 修正错误与不正确 `Date` 和 `DateTime` 比较。 #4237 (valexey)
- 修正错误的结果时 `Date` 和 `DateTime` 参数用于条件运算符（函数）的分支 `if`. 增加了函数的通用案例 `if`. #4243 (阿列克谢-米洛维多夫)

碌莽禄, 拢, 010-68520682\<url>

新功能

- 自定义每列压缩编解码器的表。 #3899 #4111 (阿利沙平, 张冬, 阿纳托利)
- 添加压缩编解码器 `Delta`. #4052 (阿利沙平)
- 允许 `ALTER` 压缩编解码器 `ecs`。 #4054 (阿利沙平)
- 新增功能 `left`, `right`, `trim`, `ltrim`, `rtrim`, `timestampadd`, `timestampsub` 对于 SQL 标准的兼容性。 #3826 (伊万·布林科夫)
- 支持写入 `HDFS` 表和 `hdfs` 表功能。 #4084 (阿利沙平)
- 增加了从 `big haystack` 中搜索多个常量字符串的功能: `multiPosition`, `multiSearch`, `firstMatch` 也与 `-UTF8`, `-CaseInsensitive`, 和 `-CaseInsensitiveUTF8` 变体。 #4053 (Danila Kutenin)
- 修剪未使用的碎片，如果 `SELECT` 通过分片键查询过滤器（设置 `optimize_skip_unused_shards`）。 #3851 (Gleb Kanterov, 伊万)
- 允许 `Kafka` 引擎忽略每个块的解析错误数。 #4094 (伊万)
- 增加了对 `CatBoost` 多类模型评估。功能 `modelEvaluate` 返回带有多类模型的每类原始预测的元组。
`libcatboostmodel.so` 应建立与 #607. #3959 (KochetovNicolai)
- 新增功能 `filesystemAvailable`, `filesystemFree`, `filesystemCapacity`. #4097 (Boris Granveaud)
- 添加了哈希函数 `xxHash64` 和 `xxHash32`. #3905 (filimonov)
- 已添加 `gccMurmurHash` 散列函数（GCC 风味杂音散列），它使用相同的散列种子 海湾合作委员会 #4000 (sundyl)

- 添加了哈希函数 `javaHash`, `hiveHash`. #3811 (上书结365)
- 添加表功能 `remoteSecure`. 函数的工作原理为 `remote`, 但使用安全连接。#4088 (proller)

实验特点

- 添加了多个联接仿真 (`allow_experimental_multiple_joins_emulation` 设置) 。#3946 (Artem Zuikov)

错误修复

- 略眉露>> `compiled_expression_cache_size` 默认情况下设置有限, 以降低内存消耗。#4041 (阿利沙平)
- 修复导致执行更改复制表的线程和从ZooKeeper更新配置的线程中挂断的错误。#2947 #3891 #3934 (Alex Zatelepin)
- 修复了执行分布式ALTER任务时的争用条件。争用条件导致多个副本试图执行任务和所有副本, 除了一个失败与 ZooKeeper错误。#3904 (Alex Zatelepin)
- 修复错误时 `from_zk` 在对ZooKeeper的请求超时后, 配置元素没有刷新。#2947 #3947 (Alex Zatelepin)
- 修复IPv4子网掩码错误前缀的错误。#3945 (阿利沙平)
- 固定崩溃 (`std::terminate`) 在极少数情况下, 由于资源耗尽而无法创建新线程。#3956 (阿列克谢-米洛维多夫)
- 修正错误时 `remote` 表函数执行时, 错误的限制被用于 `getStructureOfRemoteTable`. #4009 (阿利沙平)
- 修复netlink套接字的泄漏。它们被放置在一个池中, 在那里它们永远不会被删除, 并且当所有当前套接字都在使用时, 在新线程开始时创建了新的套接字。#4017 (Alex Zatelepin)
- 修复关闭错误 `/proc/self/fd` 目录早于所有 `fds` 被读取 `/proc` 分叉后 `odbc-bridge` 子进程。#4120 (阿利沙平)
- 在主键中使用字符串的情况下, 固定字符串到 UInt单调转换。#3870 (张冬)
- 整数转换函数单调性计算中的固定误差。#3921 (阿列克谢-米洛维多夫)
- 修复段错误 `arrayEnumerateUniq`, `arrayEnumerateDense` 函数在一些无效的参数的情况下。#3909 (阿列克谢-米洛维多夫)
- 在StorageMerge修复UB。#3910 (阿莫斯鸟)
- 修正函数中的段错误 `addDays`, `subtractDays`. #3913 (阿列克谢-米洛维多夫)
- 修正错误: 功能 `round`, `floor`, `trunc`, `ceil` 在整数参数和大负比例执行时可能会返回虚假结果。#3914 (阿列克谢-米洛维多夫)
- 修正了一个错误引起的 ‘kill query sync’ 从而导致核心转储。#3916 (muVulDeePecker)
- 修复空复制队列后延迟较长的bug。#3928 #3932 (阿利沙平)
- 修复了插入到表中的过多内存使用情况 `LowCardinality` 主键。#3955 (KochetovNicolai)
- 固定 `LowCardinality` 序列化 `Native` 在空数组的情况下格式化。#3907 #4011 (KochetovNicolai)
- 固定不正确的结果, 而使用 `distinct` 通过单 `LowCardinality` 数字列。#3895 #4012 (KochetovNicolai)
- 固定专门的聚合与 `LowCardinality` 键 (以防万一 `compile` 设置已启用)。#3886 (KochetovNicolai)
- 修复复制表查询的用户和密码转发。#3957 (阿利沙平) (小路)
- 修复了在重新加载字典时在字典数据库中列出表时可能发生的非常罕见的争用条件。#3970 (阿列克谢-米洛维多夫)
- 修正了与ROLLUP或CUBE一起使用时的错误结果。#3756 #3837 (周三)
- 用于查询的固定列别名 `JOIN ON` 语法和分布式表。#3980 (张冬)
- 在内部实现固定的错误 `quantileTDigest` (由阿尔乔姆Vakhrushev发现)。这个错误从来没有发生在ClickHouse中, 只有那些直接使用ClickHouse代码库作为库的人才有关。#3935 (阿列克谢-米洛维多夫)

改进

- 支持 `IF NOT EXISTS` 在 `ALTER TABLE ADD COLUMN` 发言以及 `IF EXISTS` 在 `DROP/MODIFY/CLEAR/COMMENT COLUMN`. #3900 (Boris Granveaud)
- 功能 `parseDateTimeBestEffort`: 支持格式 `DD.MM.YYYY`, `DD.MM.YY`, `DD-MM-YYYY`, `DD-Mon-YYYY`, `DD/Month/YYYY` 和相似。#3922 (阿列克谢-米洛维多夫)
- `CapnProtoInputStream` 现在支持锯齿结构。#4063 (Odin Hultgren Van Der Horst)
- 可用性改进: 增加了从数据目录的所有者启动服务器进程的检查。如果数据属于非root用户, 则不允许从root用户启动服务器。#3785 (谢尔盖-v-加尔采夫)
- 在分析具有联接的查询期间检查所需列的更好的逻辑。#3930 (Artem Zuikov)
- 减少在单个服务器中有大量分布式表的情况下连接数。#3726 (张冬)
- 支持的总计行 `WITH TOTALS` 查询ODBC驱动程序。#3836 (Maksim Koritckiy)
- 允许使用 `Enums` 为if函数内的整数。#3875 (伊万)
- 已添加 `low_cardinality_allow_in_native_format` 设置。如果禁用, 请不要使用 `LowCardinality` 输入 `Native` 格式。#3879 (KochetovNicolai)

- 从编译表达式缓存中删除了一些冗余对象以降低内存使用率。 #4042 (阿利沙平)
- 添加检查 `SET send_logs_level = 'value'` 查询接受适当的值。 #3873 (Sabyanin马克西姆)
- 固定数据类型检查类型转换功能。 #3896 (张冬)

性能改进

- 添加MergeTree设置 `use_minimalistic_part_header_in_zookeeper`. 如果启用，复制的表将在单个零件znode中存储紧凑零件元数据。这可以显着减少ZooKeeper快照大小（特别是如果表有很多列）。请注意，启用此设置后，您将无法降级到不支持它的版本。 #3960 (Alex Zatelepin)
- 为函数添加基于DFA的实现 `sequenceMatch` 和 `sequenceCount` 以防模式不包含时间。 #4004 (Léo Ercolanelli)
- 整数序列化的性能改进。 #3968 (阿莫斯鸟)
- 零左填充PODArray，使-1元素始终有效并归零。它用于无分支计算偏移量。 #3920 (阿莫斯鸟)
- 还原 `jemalloc` 版本导致性能下降。 #4018 (阿列克谢-米洛维多夫)

向后不兼容的更改

- 删除无证功能 `ALTER MODIFY PRIMARY KEY` 因为它被 `ALTER MODIFY ORDER BY` 指挥部 #3887 (Alex Zatelepin)
- 删除功能 `shardByHash`. #3833 (阿列克谢-米洛维多夫)
- 禁止使用具有结果类型的标量子查询 `AggregateFunction`. #3865 (伊万)

构建/测试/打包改进

- 增加了对PowerPC的支持 (`ppc64le`) 建设。 #4132 (Danila Kutenin)
- 有状态功能测试在公共可用数据集上运行。 #3969 (阿列克谢-米洛维多夫)
- 修复了服务器无法启动时的错误 `bash: /usr/bin/clickhouse-extract-from-config: Operation not permitted Docker` 或 `systemd-nspawn` 中的消息。 #4136 (阿列克谢-米洛维多夫)
- 更新 `rdkafka` 库 v1.0.0-RC5。使用 `cppkafka` 而不是原始的C接口。 #4025 (伊万)
- 更新 `mariadb-client` 图书馆。修复了UBSan发现的问题之一。 #3924 (阿列克谢-米洛维多夫)
- UBSan版本的一些修复。 #3926 #3021 #3948 (阿列克谢-米洛维多夫)
- 增加了使用UBSan构建的每次提交运行的测试。
- 增加了PVS-Studio静态分析器的每次提交运行。
- 修复了PVS-Studio发现的错误。 #4013 (阿列克谢-米洛维多夫)
- 修正了glibc兼容性问题。 #4100 (阿列克谢-米洛维多夫)
- 将Docker映像移动到18.10并为 `glibc >= 2.28` 添加兼容性文件 #3965 (阿利沙平)
- 如果用户不想在服务器码头镜像中播放目录，请添加 `env` 变量。 #3967 (阿利沙平)
- 启用了大多数来自警告 `-Weverything` 在叮当声。已启用 `-Wpedantic`. #3986 (阿列克谢-米洛维多夫)
- 增加了一些只在clang8中可用的警告。 #3993 (阿列克谢-米洛维多夫)
- 链接到 `libLLVM` 在使用共享链接时，而不是单独的LLVM库。 #3989 (Origej Desh)
- 为测试图像添加了消毒变量。 #4072 (阿利沙平)
- `clickhouse-server debian` 软件包会推荐 `libcap2-bin` 使用包 `setcap` 设置功能的工具。这是可选的。 #4093 (阿列克谢-米洛维多夫)
- 改进的编译时间，固定包括。 #3898 (proller)
- 添加了哈希函数的性能测试。 #3918 (filimonov)
- 固定循环库依赖。 #3958 (proller)
- 改进的编译与低可用内存。 #4030 (proller)
- 添加了测试脚本，以重现性能下降 `jemalloc`. #4036 (阿列克谢-米洛维多夫)
- 修正了在下面的注释和字符串文字拼写错误 `dbms`. #4122 (maiha)
- 修正了错别字的评论。 #4089 (Evgenii Pravda)

2018年的更新日志

ClickHouse释放18.16

碌莽祿,拢,010-68520682\<url>

错误修复：

- 修复了导致使用ODBC源更新字典时出现问题的错误。 #3825, #3829

- 聚集函数的JIT编译现在适用于低心率列。 #3838

改进:

- 添加了 `low_cardinality_allow_in_native_format` 设置（默认情况下启用）。如果禁用，则选择查询的LowCardinality列将转换为普通列，插入查询将需要普通列。 #3879

构建改进:

- 修复了基于macOS和ARM的构建。

碌莽禄,拢,010-68520682\<url>

新功能:

- `DEFAULT` 在以半结构化输入格式加载数据时，会计算表达式是否缺少字段 (`JSONEachRow`, `TSKV`)。该功能与启用 `insert_sample_with_metadata` 设置。 #3555
- 该 `ALTER TABLE` 查询现在有 `MODIFY ORDER BY` 用于在添加或删除表列时更改排序键的操作。这是在表有用 `MergeTree` 基于此排序键合并时执行其他任务的系列，例如 `SummingMergeTree`, `AggregatingMergeTree`，等等。 #3581 #3755
- 对于在表 `MergeTree` 家庭，现在你可以指定一个不同的排序键 (`ORDER BY`) 和索引 (`PRIMARY KEY`)。排序键可以长于索引。 #3581
- 添加了 `hdfs` 表功能和 `HDFS` 用于将数据导入和导出到HDFS的表引擎。晨兴-xc
- 增加了使用`base64`的功能: `base64Encode`, `base64Decode`, `tryBase64Decode`. Alexander Krasheninnikov
- 现在，您可以使用一个参数来配置的精度 `uniqCombined` 聚合函数（选择HyperLogLog单元格的数量）。 #3406
- 添加了 `system.contributors` 包含在ClickHouse中进行提交的所有人的名称的表。 #3452
- 增加了省略分区的能力 `ALTER TABLE ... FREEZE` 查询以便一次备份所有分区。 #3514
- 已添加 `dictGet` 和 `dictGetOrDefault` 不需要指定返回值类型的函数。该类型是从字典描述自动确定的。阿莫斯鸟
- 现在，您可以在表描述中为列指定注释，并使用以下方式对其进行更改 `ALTER`. #3377
- 阅读支持 `Join` 使用简单键键入表格。阿莫斯鸟
- 现在，您可以指定选项 `join_use_nulls`, `max_rows_in_join`, `max_bytes_in_join`，和 `join_overflow_mode` 当创建一个 `Join` 键入表。阿莫斯鸟
- 添加了 `joinGet` 功能，允许您使用 `Join` 像字典一样键入表格。阿莫斯鸟
- 添加了 `partition_key`, `sorting_key`, `primary_key`，和 `sampling_key` 列到 `system.tables` 表以便提供关于表键的信息。 #3609
- 添加了 `is_in_partition_key`, `is_in_sorting_key`, `is_in_primary_key`，和 `is_in_sampling_key` 列到 `system.columns` 桌子 #3609
- 添加了 `min_time` 和 `max_time` 列到 `system.parts` 桌子 当分区键是由以下表达式组成的表达式时，将填充这些列 `DateTime` 列。Emmanuel Donin de Rosière

错误修复:

- 修复和性能改进 `LowCardinality` 数据类型。 `GROUP BY` 使用 `LowCardinality(Nullable(...))`。获取的值 `extremes`. 处理高阶函数。 `LEFT ARRAY JOIN`. 分布 `GROUP BY`. 返回的函数 `Array`. 执行 `ORDER BY`. 写入 `Distributed` 表 (`nicelulu`)。向后兼容 `INSERT` 从实现旧客户端的查询 Native 协议 支持 `LowCardinality` 为 `JOIN`. 在单个流中工作时提高性能。 #3823 #3803 #3799 #3769 #3744 #3681 #3651 #3649 #3641 #3632 #3568 #3523 #3518
- 固定如何 `select_sequential_consistency` 选项工作。以前，启用此设置时，在开始写入新分区后，有时会返回不完整的结果。 #2863
- 执行DDL时正确指定数据库 `ON CLUSTER` 查询和 `ALTER UPDATE/DELETE`. #3772 #3460
- 为视图中的子查询正确指定了数据库。 #3521
- 修正了一个错误 `PREWHERE` 与 `FINAL` 为 `VersionedCollapsingMergeTree`. 7167bfd7
- 现在你可以使用 `KILL QUERY` 取消尚未启动的查询，因为它们正在等待锁定表。 #3517
- 更正日期和时间计算，如果时钟被移回午夜（这发生在伊朗，并发生在莫斯科1981年至1983年）。以前，这导致时间比必要的时早一天重置，并且还导致文本格式的日期和时间格式不正确。 #3819
- 修正了某些情况下的错误 `VIEW` 和省略数据库的子查询。张冬
- 修正了一个争用条件时，同时从读取 `MATERIALIZED VIEW` 和删除 `MATERIALIZED VIEW` 由于不锁定内部 `MATERIALIZED VIEW`. #3404 #3694
- 修正了错误 `Lock handler cannot be nullptr`. #3689

- 固定查询处理时 `compile_expressions` 选项已启用（默认情况下启用）。非确定性常量表达式，如 `now` 功能不再展开。#3457
- 修复了在指定非常量比例参数时发生的崩溃 `toDecimal32/64/128` 功能。
- 修复了尝试插入数组时的错误 `NULL` 中的元素 `Values` 格式化为类型的列 `Array` 没有 `Nullable` （如果 `input_format_values_interpret_expressions = 1`）。#3487 #3503
- 固定连续错误登录 `DDLWorker` 如果动物园管理员不可用。8f50c620
- 修正了返回类型 `quantile*` 从功能 `Date` 和 `DateTime` 参数的类型。#3580
- 修正了 `WITH` 子句，如果它指定了一个没有表达式的简单别名。#3570
- 固定处理具有命名子查询和限定列名的查询时 `enable_optimize_predicate_expression` 被启用。张冬
- 修正了错误 `Attempt to attach to nullptr thread group` 使用实例化视图时。Marek Vavruša
- 修正了传递某些不正确的参数时崩溃 `arrayReverse` 功能。73e3a7b6
- 修正了缓冲区溢出 `extractURLParameter` 功能。改进的性能。添加了包含零字节的字符串的正确处理。141e9799
- 在固定缓冲区溢出 `lowerUTF8` 和 `upperUTF8` 功能。删除了执行这些功能的能力 `FixedString` 类型参数。#3662
- 修复了删除时罕见的竞争条件 `MergeTree` 桌子 #3680
- 修正了从读取时的争用条件 `Buffer` 表和同时执行 `ALTER` 或 `DROP` 在目标桌上。#3719
- 修正了一个段错误，如果 `max_temporary_non_const_columns` 超过限制。#3788

改进：

- 服务器不会将处理后的配置文件写入 `/etc/clickhouse-server/` 目录。相反，它将它们保存在 `preprocessed_configs` 里面的目录 `path`。这意味着 `/etc/clickhouse-server/` 目录没有写访问权限 `clickhouse` 用户，从而提高了安全性。#2443
- 该 `min_merge_bytes_to_use_direct_io` 默认情况下，选项设置为 10GiB。将在 `MergeTree` 系列中执行形成大部分表的合并 `O_DIRECT` 模式，这可以防止过多的页高速缓存逐出。#3504
- 当表数量非常多时，加速服务器启动。#3398
- 添加了连接池和 HTTP `Keep-Alive` 用于副本之间的连接。#3594
- 如果查询语法无效，则 400 Bad Request 代码在返回 HTTP 接口（500 以前返回）。31bc680a
- 该 `join_default_strictness` 选项设置为 `ALL` 默认情况下为兼容性。120e2cbe
- 删除日志记录 `stderr` 从 `re2` 无效或复杂正则表达式的库。#3723
- 添加的 Kafka 表引擎：在开始从 Kafka 读取之前检查订阅；表的 `kafka_max_block_size` 设置。Marek Vavruša
- 该 `cityHash64`, `farmHash64`, `metroHash64`, `sipHash64`, `halfMD5`, `murmurHash2_32`, `murmurHash2_64`, `murmurHash3_32`，和 `murmurHash3_64` 函数现在适用于任意数量的参数和元组形式的参数。#3451 #3519
- 该 `arrayReverse` 函数现在适用于任何类型的数组。73e3a7b6
- 增加了一个可选参数：插槽大小的 `timeSlots` 功能。基里尔·什瓦科夫
- 为 `FULL` 和 `RIGHT JOIN`，该 `max_block_size` 设置用于右表中未连接的数据流。阿莫斯鸟
- 添加了 `--secure` 命令行参数 `clickhouse-benchmark` 和 `clickhouse-performance-test` 启用 TLS。#3688 #3690
- 类型转换时的结构 `Buffer` 表的类型与目标表的结构不匹配。维塔利·巴拉诺夫
- 添加了 `tcp_keep_alive_timeout` 在指定的时间间隔内不活动后启用保持活动数据包的选项。#3441
- 删除不必要的引用值的分区键中 `system.parts` 表，如果它由单列组成。#3652
- 模函数适用于 `Date` 和 `DateTime` 数据类型。#3385
- 添加同义词的 `POWER`, `LN`, `LCASE`, `UCASE`, `REPLACE`, `LOCATE`, `SUBSTR`，和 `MID` 功能。#3774 #3763 为了与 SQL 标准兼容，某些函数名称不区分大小写。添加语法糖 `SUBSTRING(expr FROM start FOR length)` 对于与 SQL 的兼容性。#3804
- 增加了以下能力 `mlock` 对应于存储器页 `clickhouse-server` 可执行代码，以防止它被强制出内存。默认情况下禁用此功能。#3553
- 从读取时改进的性能 `O_DIRECT`（与 `min_bytes_to_use_direct_io` 选项启用）。#3405
- 的改进的性能 `dictGet...OrDefault` 常量键参数和非常量默认参数的函数。阿莫斯鸟
- 该 `firstSignificantSubdomain` 功能现在处理域 `gov`, `mil`，和 `edu`. Igor Hatarist 改进的性能。#3628
- 能够指定用于启动的自定义环境变量 `clickhouse-server` 使用 `SYS-V init.d` 通过定义脚本 `CLICKHOUSE_PROGRAM_ENV` 在 `/etc/default/clickhouse`.

Pavlo Bashynskyi

- Clickhouse-server init 脚本的正确返回代码。#3516
- 该 `system.metrics` 表现在有 `VersionInteger` 公制和 `system.build_options` 有添加的行 `VERSION_INTEGER`，其中包含 ClickHouse 版本的数字形式，例如 18016000. #3644
- 删除比较的能力 `Date` 输入一个数字，以避免潜在的错误，如 `date = 2018-12-17`，其中日期周围的引号被错误省略。#3687
- 修正了古语大忌数的行为。仙人们前给山的什么且山下大木沟八折期间户动工土一人私字。

- 修正了有嵌套函数的插入，如 `TOWNUMBERINALLBLOCKS`。他们之前输出的结果不是由一个空格分隔的，而是由一个逗号分隔的。 阿莫斯鸟
- 如果 `force_restore_data` 文件无法删除，将显示错误消息。 阿莫斯鸟

构建改进：

- 更新了 `jemalloc` 库，它修复了潜在的内存泄漏。 阿莫斯鸟
- 分析与 `jemalloc` 默认情况下为了调试生成启用。 `2cc82f5c`
- 增加了运行集成测试的能力，当只 `Docker` 安装在系统上。 [#3650](#)
- 在 `SELECT` 查询中添加了模糊表达式测试。 [#3442](#)
- 为提交添加了一个压力测试，它以并行和随机顺序执行功能测试，以检测更多的竞争条件。 [#3438](#)
- 改进了在 `Docker` 映像中启动 `clickhouse-server` 的方法。 [Elghazal Ahmed](#)
- 对于 `Docker` 映像，增加了对使用数据库中的文件初始化数据库的支持 `/docker-entrypoint-initdb.d` 目录。 [康斯坦丁·列别杰夫](#)
- 修复了基于 ARM 的构建。 [#3709](#)

向后不兼容的更改：

- 删除比较的能力 `Date` 用数字键入。而不是 `toDate('2018-12-18') = 17883`，必须使用显式类型转换 `= toDate(17883)` [#3687](#)

ClickHouse 释放 18.14

碌莽禄, 拢, 010-68520682\<url>

错误修复：

- 修复了导致使用 ODBC 源更新字典时出现问题的错误。 [#3825, #3829](#)
- 执行 DDL 时正确指定数据库 `ON CLUSTER` 查询。 [#3460](#)
- 修正了一个段错误，如果 `max_temporary_non_const_columns` 超过限制。 [#3788](#)

构建改进：

- 修复了基于 ARM 的构建。

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复：

- 修正错误 `dictGet...` 类型字典的函数 `range`，如果其中一个参数是恒定的，而另一个则不是。 [#3751](#)
- 修复了导致消息的错误 `netlink: '...': attribute type 1 has an invalid length` 要打印在 Linux 内核日志中，这发生在足够新鲜的 Linux 内核版本上。 [#3749](#)
- 在功能固定段错误 `empty` 对于争论 `FixedString` 类型。 [丹尼尔, 道广明](#)
- 修正了使用大值时过多的内存分配 `max_query_size` 设置（内存块 `max_query_size` 字节被预先分配一次）。 [#3720](#)

构建更改：

- 使用操作系统包中的版本 7 的 LLVM/Clang 库修复构建（这些库用于运行时查询编译）。 [#3582](#)

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复：

- 修复了 ODBC 桥进程未与主服务器进程终止的情况。 [#3642](#)
- 固定同步插入 `Distributed` 具有不同于远程表的列列表的表。 [#3673](#)
- 修复了丢弃 MergeTree 表时可能导致崩溃的罕见竞争条件。 [#3643](#)
- 修复了查询线程创建失败时的查询死锁 `Resource temporarily unavailable` 错误 [#3643](#)
- 修正了解析 `ENGINE` 条款时 `CREATE AS table` 语法被使用和 `ENGINE` 子句之前指定 `AS table`（错误导致忽略指定的引擎）。 [#3692](#)

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复：

- 反序列化类型的列时，高估了内存块的大小 `Array(String)` 这导致“Memory limit exceeded”错误。该问题出现在版

本18.12.13中。 #3589

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复:

- 固定 ON CLUSTER 当群集配置为安全时进行查询 (标志 <secure>). #3599

构建更改:

- 固定的问题 (llvm-7从系统, macos) #3582

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复:

- 修正了 Block structure mismatch in MergingSorted stream 错误 #3162
- 固定 ON CLUSTER 查询的情况下, 当安全连接被打开的群集配置 (<secure> 标志) 。 #3465
- 修复了查询中使用的错误 SAMPLE, PREWHERE 和别名列。 #3543
- 修正了一个罕见的 unknown compression method 错误时 min_bytes_to_use_direct_io 设置已启用。 3544

性能改进:

- 查询的固定性能回归 GROUP BY 在AMD EPYC处理器上执行时, uint16或Date类型的列。 Igor Lapko
- 修正了处理长字符串的查询的性能回归。 #3530

构建改进:

- 简化阿卡迪亚构建的改进。 #3475, #3535

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复:

- 修复了加入两个未命名的子查询时的崩溃。 #3505
- 修正了生成不正确的查询 (用空 WHERE 子句) 查询外部数据库时。 hotid
- 修正了在ODBC字典中使用不正确的超时值。 Marek Vavruša

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

错误修复:

- 修正了错误 Block structure mismatch in UNION stream: different number of columns 在限制查询。 #2156
- 修复了在嵌套结构中包含数组的表中合并数据时出现的错误。 #3397
- 修正了不正确的查询结果, 如果 merge_tree_uniform_read_distribution 设置被禁用 (默认情况下启用) 。 #3429
- 修复了在本机格式的分布式表中插入错误。 #3411

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

- 该 compile_expressions 默认情况下禁用设置 (表达式JIT编译) 。 #3410
- 该 enable_optimize_predicate_expression 默认情况下禁用设置。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

- 该 WITH CUBE 修饰符 GROUP BY (替代语法 GROUP BY CUBE(...) 也可用) 。 #3172
- 添加了 formatDateTime 功能。 [Alexandr Krasheninnikov](#)
- 添加了 JDBC 表引擎和 jdbc 表功能 (需要安装clickhouse-jdbc桥) 。 [Alexandr Krasheninnikov](#)
- 增加了使用ISO周编号的功能: toISOWeek, toISOYear, toStartOfISOYear, 和 toDayOfYear. #3146
- 现在你可以使用 Nullable 列 MySQL 和 ODBC 桌子 #3362
- 嵌套的数据结构可以被读取为嵌套的对象 JSONEachRow 格式。添加了 input_format_import_nested_json 设置。 [维罗曼*云坎](#)
- 并行处理可用于许多 MATERIALIZED VIEWS 插入数据时。见 parallel_view_processing 设置。 [Marek Vavruša](#)
- 添加了 SYSTEM FLUSH LOGS 查询 (强制日志刷新到系统表, 如 query_log) #3321
- 现在, 您可以使用预定义 database 和 table 声明时的宏 Replicated 桌子 #3251

- 增加了阅读的能力 `Decimal` 工程表示法中的类型值（表示十的幂）。#3153

实验特点：

- 对 GROUP BY 子句进行优化 LowCardinality data types. #3138
- 表达式的优化计算 LowCardinality data types. #3200

改进：

- 显着减少查询的内存消耗 `ORDER BY` 和 `LIMIT`. 见 `max_bytes_before_remerge_sort` 设置。#3205
- 在没有 `JOIN (LEFT, INNER, ...)`, `INNER JOIN` 是假定的。#3147
- 限定星号在以下查询中正常工作 `JOIN`. 张冬
- 该 ODBC 表引擎正确地选择用于引用远程数据库的SQL方言中的标识符的方法。Alexandr Krasheninnikov
- 该 `compile_expressions` 默认情况下启用设置（表达式的JIT编译）。
- 修复了同时删除数据库/表（如果存在）和创建数据库/表（如果不存在）的行为。前情提要 `CREATE DATABASE ... IF NOT EXISTS` 查询可能会返回错误消息 “File ... already exists” 和 `CREATE TABLE ... IF NOT EXISTS` 和 `DROP TABLE IF EXISTS` 查询可能会返回 Table ... is creating or attaching right now. #3101
- 当从MySQL或ODBC表中查询时，`LIKE`和`IN`表达式具有常量右半部分被传递到远程服务器。#3182
- 当从MySQL和ODBC表查询时，与`WHERE`子句中常量表达式的比较会传递给远程服务器。以前，只通过与常量的比较。#3182
- 正确计算终端中的行宽 `Pretty` 格式，包括带有象形文字的字符串。阿莫斯鸟。
- `ON CLUSTER` 可以指定 `ALTER UPDATE` 查询。
- 提高了读取数据的性能 `JSONEachRow` 格式。#3332
- 添加同义词的 `LENGTH` 和 `CHARACTER_LENGTH` 功能的兼容性。该 `CONCAT` 函数不再区分大小写。#3306
- 添加了 `TIMESTAMP` 的同义词 `DateTime` 类型。#3390
- 服务器日志中始终为 `query_id` 保留空间，即使日志行与查询无关。这使得使用第三方工具更容易分析服务器文本日志。
- 当查询超过整数千兆字节的下一级别时，会记录查询的内存消耗。#3205
- 为使用本机协议的客户端库错误发送的列少于服务器预期的插入查询时的情况添加了兼容模式。使用clickhouse-cpp 库时，这种情况是可能的。以前，此方案会导致服务器崩溃。#3171
- 在用户定义的`WHERE`表达式中 `clickhouse-copier`，您现在可以使用 `partition_key` 别名（用于按源表分区进行其他过滤）。如果分区方案在复制过程中发生更改，但仅稍有更改，这很有用。#3166
- 的工作流程 Kafka 引擎已被移动到后台线程池中，以便在高负载下自动降低数据读取速度。Marek Vavruša.
- 支持阅读 `Tuple` 和 `Nested` 结构的值，如 `struct` 在 Cap'n'Proto format. Marek Vavruša
- 顶级域名列表 `firstSignificantSubdomain` 功能现在包括域 `biz`. decaseal
- 在外部字典的配置，`null_value` 被解释为默认数据类型的值。#3330
- 支持 `intDiv` 和 `intDivOrZero` 功能 `Decimal`. b48402e8
- 支持 `Date`, `DateTime`, `UUID`，和 `Decimal` 类型作为键 `sumMap` 聚合函数。#3281
- 支持 `Decimal` 外部字典中的数据类型。#3324
- 支持 `Decimal` 数据类型 in `SummingMergeTree` 桌子 #3348
- 增加了专业化 `UUID` 在 `if`. #3366
- 减少的数量 `open` 和 `close` 从读取时系统调用 `MergeTree table`. #3283
- A `TRUNCATE TABLE` 查询可以在任何副本上执行（将查询传递给领导副本）。基里尔*什瓦科夫

错误修复：

- 修正了一个问题 `Dictionary` 表 `range_hashed` 字典 此错误发生在版本18.12.17中。#1702
- 修正了加载时的错误 `range_hashed` 字典（消息 `Unsupported type Nullable (...)`）。此错误发生在版本18.12.17中。#3362
- 在固定的错误 `pointInPolygon` 函数由于不准确的计算的多边形与大量的顶点位于彼此靠近的积累。#3331 #3341
- 如果在合并数据部分之后，结果部分的校验和与另一个副本中相同合并的结果不同，则删除合并的结果并从另一个副本下载数据部分（这是正确的行为）。但是在下载数据部分之后，由于该部分已经存在的错误（因为合并后数据部分被删除了一些延迟），因此无法将其添加到工作集中。这导致周期性尝试下载相同的数据。#3194
- 修正了查询总内存消耗的不正确计算（由于计算不正确， `max_memory_usage_for_all_queries` 设置工作不正确， `MemoryTracking` 度量值不正确）。此错误发生在版本18.12.13中。Marek Vavruša
- 修正的功能 `CREATE TABLE ... ON CLUSTER ... AS SELECT ...` 此错误发生在版本18.12.13中。#3247
- 修正了数据结构的不必要的准备 `JOIN`如果发起查询的服务器上 `JOIN` 仅在远程服务器上执行。#3340
- 在固定的错误 Kafka 引擎：开始读取数据时异常后的死锁，并在完成时锁定 Marek Vavruša

- 为 Kafka 表，可选 schema 参数未被传递（的架构 Cap'n'Proto 格式）。Vojtech Splichal
- 如果 ZooKeeper 服务器的整体接受连接，但随后立即关闭它，而不是响应握手，ClickHouse 选择连接另一台服务器。以前，这会产生错误 Cannot read all data. Bytes read: 0. Bytes expected: 4. 服务器无法启动。8218cf3a
- 如果 ZooKeeper 服务器的整体包含 DNS 查询返回错误的服务器，则忽略这些服务器。17b8e209
- 固定类型之间的转换 Date 和 DateTime 当在插入数据 VALUES 格式（如果 input_format_values_interpret_expressions = 1）。以前，转换是在 Unix Epoch 时间中的天数和 Unix 时间戳的数值之间进行的，这会导致意外的结果。#3229
- 修正类型之间的转换 Decimal 和整数。#3211
- 在固定的错误 enable_optimize_predicate_expression 设置。张冬
- 如果使用非默认的 CSV 分隔符，则修复了 CSV 格式的浮点数解析错误，例如；#3155
- 修正了 arrayCumSumNonNegative 函数（它不累加负值，如果累加器小于零）。Aleksey Studnev
- 固定如何 Merge 表工作的顶部 Distributed 使用时的表 PREWHERE. #3165
- 在错误修复 ALTER UPDATE 查询。
- 在固定的错误 odbc 表功能，出现在版本 18.12。#3197
- 修正了聚合函数的操作 StateArray 组合子 #3188
- 修正了划分时崩溃 Decimal 值为零。69dd6609
- 使用固定输出类型的操作 Decimal 和整数参数。#3224
- 修正了在段错误 GROUP BY 上 Decimal128. 3359ba06
- 该 log_query_threads 设置（关于查询执行的每个线程的日志记录信息）现在生效，只有当 log_queries 选项（有关查询的日志记录信息）设置为 1。由于 log_query_threads 默认情况下，即使禁用了查询日志记录，也会先前记录有关线程的信息。#3241
- 修正了分位数聚合函数的分布式操作中的错误（错误消息 Not found column quantile...）。292a8855
- 修复了同时在 18.12.17 版服务器和旧服务器的集群上工作时的兼容性问题。对于具有固定和非固定长度的 GROUP BY 键的分布式查询，如果要聚合大量数据，则返回的数据并不总是完全聚合（两个不同的行包含相同的聚合键）。#3254
- 固定处理替换 clickhouse-performance-test，如果查询只包含测试中声明的替换的一部分。#3263
- 修复了使用时的错误 FINAL 与 PREWHERE. #3298
- 修复了使用时的错误 PREWHERE 在过程中添加的列 ALTER. #3298
- 增加了一个检查没有 arrayJoin 为 DEFAULT 和 MATERIALIZED 表达式。前情提要，arrayJoin 插入数据时导致错误。#3337
- 增加了一个检查没有 arrayJoin 在一个 PREWHERE 条款 以前，这导致了类似的消息 Size ... doesn't match 或 Unknown compression method 执行查询时。#3357
- 修复了优化后可能发生的极少数情况下的段错误，并将相等性评估与相应的 IN 表达式链接起来。刘一民-字节舞
- 小幅更正 clickhouse-benchmark：以前，客户端信息没有发送到服务器；现在关闭时更准确地计算执行的查询数量，并限制迭代次数。#3351 #3352

向后不兼容的更改：

- 删除了 allow_experimental_decimal_type 选项。该 Decimal 数据类型可供默认使用。#3329

ClickHouse 释放 18.12

碌莽碌, 拢, 010-68520682\<url>

新功能：

- invalidate_query（指定查询来检查是否需要更新外部字典的能力）实现了 clickhouse 资料来源。#3126
- 增加了使用的能力 UInt*, Int*，和 DateTime 数据类型（与 Date 类型）作为 range_hashed 定义范围边界的外部字典键。现在 NULL 可用于指定开放范围。瓦西里*内姆科夫
- 该 Decimal 类型现在支持 var* 和 stddev* 聚合函数。#3129
- 该 Decimal 类型现在支持数学函数 (exp, sin 等等。) #3129
- 该 system.part_log 表现在有 partition_id 列。#3089

错误修复：

- Merge 现在正常工作 Distributed 桌子 张冬
- 修复了不兼容（不必要的依赖 glibc 版本），这使得它不可能运行 ClickHouse 的 Ubuntu Precise 和旧版本。在版本 18.12.13 中出现了不兼容。#3130

- 在固定的错误 `enable_optimize_predicate_expression` 设置。 张冬
- 修复了在早于 18.12.13 的版本上使用副本集群并同时在具有较新版本的服务器上创建表的新副本时出现的向后兼容性的一个小问题（如消息中所示 `Can not clone replica, because the ... updated to new ClickHouse version`，这是合乎逻辑的，但不应该发生）。 #3122

向后不兼容的更改：

- 该 `enable_optimize_predicate_expression` 默认情况下启用选项（这是相当乐观的）。如果发生与搜索列名相关的查询分析错误，请设置 `enable_optimize_predicate_expression` 为 0。张冬

碌莽禄, 拢, 0755-88888888

新功能：

- 增加了对 `ALTER UPDATE` 查询。 #3035
- 添加了 `allow_ddl` 选项，它限制用户对 DDL 查询的访问。 #3104
- 添加了 `min_merge_bytes_to_use_direct_io` 备选案文 `MergeTree` 引擎允许您为合并的总大小设置阈值（当超过阈值时，将使用 `O_DIRECT` 处理数据部分文件）。 #3117
- 该 `system.merges` 系统表现在包含 `partition_id` 列。 #3099

改进

- 如果数据部分在变异期间保持不变，则副本不会下载该数据部分。 #3103
- 使用时，自动完成可用于设置名称 `clickhouse-client`. #3106

错误修复：

- 添加了一个检查是元素的数组的大小 `Nested` 插入时的类型字段。 #3118
- 修正了一个错误更新外部字典与 `ODBC` 来源和 `hashed` 存储。此错误发生在版本 18.12.13 中。
- 修复了使用以下命令从查询创建临时表时出现的崩溃 `IN` 条件。 张冬
- 修复了聚合函数中可能具有的数组的错误 `NULL` 元素。 张冬

碌莽禄, 拢, 010-68520682\<url>

新功能：

- 添加了 `DECIMAL(digits, scale)` 数据类型 (`Decimal32(scale)`, `Decimal64(scale)`, `Decimal128(scale)`)。要启用它，请使用以下设置 `allow_experimental_decimal_type`. #2846 #2970 #3008 #3047
- 新 `WITH ROLLUP` 修饰符 `GROUP BY` (替代语法: `GROUP BY ROLLUP(...)`). #2948
- 在具有 `JOIN` 的查询中，星形字符将扩展为符合 SQL 标准的所有表中的列列表。您可以通过设置恢复旧行为 `asterisk_left_columns_only` 在用户配置级别上为 1。 张冬
- 增加了对连接表函数的支持。 张冬
- 在 `clickhouse-client` 中按 Tab 键进行自动完成。 谢尔盖*谢尔宾
- `Clickhouse-client` 中的 Ctrl+C 清除输入的查询。 #2877
- 添加了 `join_default_strictness` 设置 (值: `", 'any', 'all'`)。这允许您不指定 `ANY` 或 `ALL` 为 `JOIN`. #2982
- 与查询处理相关的服务器日志的每一行都显示了查询 ID。 #2482
- 现在，您可以在 `clickhouse-client` 中获取查询执行日志 (使用 `send_logs_level` 设置)。通过分布式查询处理，日志从所有服务器级联。 #2482
- 该 `system.query_log` 和 `system.processes` (`SHOW PROCESSLIST`) 表现在有关所有更改的设置信息，当你运行一个查询 (的嵌套结构 `Settings` 数据)。添加了 `log_query_settings` 设置。 #2482
- 该 `system.query_log` 和 `system.processes` 表现在显示有关参与查询执行的线程数的信息 (请参阅 `thread_numbers` 列)。 #2482
- 已添加 `ProfileEvents` 用于度量通过网络读取和写入磁盘以及读取和写入磁盘所花费的时间、网络错误的数量以及在网络带宽受限时所花费的等待时间。 #2482
- 已添加 `ProfileEvents` 包含来自 `rusage` 的系统指标的计数器 (您可以使用它们获取有关用户空间和内核、页面错误和上下文切换的 CPU 使用率的信息)，以及 `taskstats` 指标 (使用它们获取有关 I/O 等待时间、CPU 等待时间以及读取和记录的数据量的信息，无论是否包含页面缓存)。 #2482
- 该 `ProfileEvents` 计数器应用于全局和每个查询，以及每个查询执行线程，它允许您按查询详细分析资源消耗情况。 #2482
- 添加了 `system.query_thread_log` 表，其中包含有关每个查询执行线程的信息。添加了 `log_query_threads` 设置。 #2482

#2482

- 该 `system.metrics` 和 `system.events` 表现在有内置文档。 #3016
- 添加了 `arrayEnumerateDense` 功能。 阿莫斯鸟
- 添加了 `arrayCumSumNonNegative` 和 `arrayDifference` 功能。 Aleksey Studnev
- 添加了 `retention` 聚合函数。 李尚迪
- 现在，您可以使用 `plus` 运算符添加（合并）聚合函数的状态，并将聚合函数的状态乘以非负常数。 #3062 #3034
- MergeTree 系列中的表现在具有虚拟列 `_partition_id`. #3089

实验特点：

- 添加了 `LowCardinality(T)` 数据类型。此数据类型自动创建值的本地字典，并允许数据处理而无需解压字典。 #2830
- 添加了 JIT 编译函数的缓存和编译前使用次数的计数器。要 JIT 编译表达式，请启用 `compile_expressions` 设置。 #2990 #3077

改进：

- 修复了放弃副本时复制日志无限积累的问题。为延迟较长的副本添加了有效的恢复模式。
- 改进的性能 `GROUP BY` 当其中一个是 `string`，其他是固定长度时，具有多个聚合字段。
- 使用时提高性能 `PREWHERE` 并与表达式的隐式转移 `PREWHERE`.
- 改进文本格式的解析性能 (`CSV, TSV`). 阿莫斯鸟 #2980
- 改进了读取二进制格式字符串和数组的性能。阿莫斯鸟
- 提高性能和减少内存消耗的查询 `system.tables` 和 `system.columns` 当单个服务器上有非常大量的表时。 #2953
- 修复了大量查询导致错误的情况下性能问题 (`_dl_addr` 功能是可见的 `perf top`，但服务器没有使用太多的CPU)。 #2938
- 条件被转换到视图中（当 `enable_optimize_predicate_expression` 被启用）。张冬
- 改进的功能 `UUID` 数据类型。 #3074 #2985
- 该 `UUID`-Alchemist 字典支持数据类型。 #2822
- 该 `visitParamExtractRaw` 函数与嵌套结构正常工作。 张冬
- 当 `input_format_skip_unknown_fields` 启用设置，在对象字段 `JSONEachRow` 格式被正确跳过。 BlahGeek
- 对于一个 `CASE` 表达式与条件，你现在可以省略 `ELSE`，这相当于 `ELSE NULL`. #2920
- 现在可以在使用 ZooKeeper 时配置操作超时。 urykhy
- 您可以指定偏移量 `LIMIT n, m` 作为 `LIMIT n OFFSET m`. #2840
- 您可以使用 `SELECT TOP n` 语法作为替代 `LIMIT`. #2840
- 增加了队列的大小写入系统表，因此 `SystemLog parameter queue is full` 错误不经常发生。
- 该 `windowFunnel aggregate` 函数现在支持满足多个条件的事件。 阿莫斯鸟
- 重复的列可以用于 `USING` 条款 `JOIN`. #3006
- Pretty 格式现在对列对齐宽度有限制。使用 `output_format_pretty_max_column_pad_width` 设置。如果一个值较宽，它仍将完整显示，但表中的其他单元格不会太宽。 #3003
- 该 `odbc` 表函数现在允许您指定数据库/模式名称。 阿莫斯鸟
- 增加了使用在指定的用户名的能力 `clickhouse-client` 配置文件。 弗拉基米尔*科兹宾
- 该 `ZooKeeperExceptions` 计数器已被分成三个计数器：`ZooKeeperUserExceptions`, `ZooKeeperHardwareExceptions`，和 `ZooKeeperOtherExceptions`.
- `ALTER DELETE` 查询适用于实例化视图。
- 在定期运行清理线程时添加了随机化 `ReplicatedMergeTree` 表，以避免周期性负载尖峰时有一个非常大的数量 `ReplicatedMergeTree` 桌子
- 支持 `ATTACH TABLE ... ON CLUSTER` 查询。 #3025

错误修复：

- 修正了一个问题 `Dictionary` 表（抛出 `Size of offsets doesn't match size of column` 或 `Unknown compression method` 例外）。此错误出现在版本 18.10.3 中。 #2913
- 修复了合并时的错误 `CollapsingMergeTree` 如果其中一个数据部分为空（这些部分在合并或合并期间形成 `ALTER DELETE` 如果所有数据被删除），和 `vertical` 算法被用于合并。 #3049
- 在固定的竞争条件 `DROP` 或 `TRUNCATE` 为 `Memory` 表与同时 `SELECT`，这可能导致服务器崩溃。此错误出现在版本 1.1.54388 中。 #3038
- 修正了插入时数据丢失的可能性 `Replicated` 表如果 `Session is expired` 错误返回（数据丢失可以通过检测 `ReplicatedDataLoss` 公制）。此错误发生在版本 1.1.54378。 #2939 #2949 #2964
- 在修复段错误 `JOIN ... ON`. #3000

- 修正了错误搜索列名时 WHERE 表达式完全由限定列名组成，例如 WHERE table.column. #2994
- 修正了“Not found column”如果从远程服务器请求由IN表达式和子查询组成的单个列，则在执行分布式查询时发生错误。 #3087
- 修正了 Block structure mismatch in UNION stream: different number of columns 如果其中一个分片是本地的，而另一个分片不是，则发生分布式查询的错误，并优化移动到 PREWHERE 被触发。 #2226 #3037 #3055 #3065 #3073 #3090 #3093
- 修正了 pointInPolygon 非凸多边形的某些情况下的函数。 #2910
- 修正了比较时不正确的结果 nan 与整数。 #3024
- 修正了一个错误 zlib-ng 在极少数情况下可能导致segfault的库。 #2854
- 修复了插入到表中时的内存泄漏 AggregateFunction 列，如果聚合函数的状态不简单（分别分配内存），并且如果单个插入请求导致多个小块。 #3084
- 修复了创建和删除相同的竞争条件 Buffer 或 MergeTree 同时表。
- 修复了比较由某些非平凡类型（如元组）组成的元组时出现段错误的可能性。 #2989
- 修正了运行某些时段错误的可能性 ON CLUSTER 查询。 张冬
- 修正了一个错误 arrayDistinct 功能 Nullable 数组元素。 #2845 #2937
- 该 enable_optimize_predicate_expression 选项现在正确支持的情况下 SELECT *. 张冬
- 修复了重新初始化ZooKeeper会话时的段错误。 #2917
- 与ZooKeeper工作时固定的潜在阻塞。
- 修正了不正确的代码添加嵌套的数据结构中 SummingMergeTree.
- 在为聚合函数的状态分配内存时，会正确考虑对齐，这使得在实现聚合函数的状态时可以使用需要对齐的操作。 晨兴-
xc

安全修复：

- 安全使用ODBC数据源。与ODBC驱动程序的交互使用单独的 clickhouse-odbc-bridge 过程。第三方ODBC驱动程序中的错误不再导致服务器稳定性问题或漏洞。 #2828 #2879 #2886 #2893 #2921
- 修正了在文件路径的不正确的验证 catBoostPool 表功能。 #2894
- 系统表的内容 (tables, databases, parts, columns, parts_columns, merges, mutations, replicas, 和 replication_queue) 根据用户对数据库的配置访问权限进行过滤 (allow_databases). 张冬

向后不兼容的更改：

- 在具有JOIN的查询中，星形字符将扩展为符合SQL标准的所有表中的列列表。您可以通过设置恢复旧行为 asterisk_left_columns_only 在用户配置级别上为1。

构建更改：

- 大多数集成测试现在可以通过commit运行。
- 代码样式检查也可以通过提交运行。
- 该 memcpy 在CentOS7/Fedora上构建时，正确选择实现。 Etienne Champetier
- 当使用clang来构建时，来自一些警告 -Weverything 已添加，除了常规 -Wall-Wextra -Werror. #2957
- 调试生成使用 jemalloc 调试选项。
- 用于与ZooKeeper交互的库接口被声明为抽象。 #2950

ClickHouse释放18.10

碌莽禄,拢,010-68520682\<url>

新功能：

- HTTPS可用于复制。 #2760
- 新增功能 murmurHash2_64, murmurHash3_32, murmurHash3_64，和 murmurHash3_128 除了现有的 murmurHash2_32. #2791
- 支持ClickHouse ODBC驱动程序中的可空类型 (ODBCDriver2 输出格式) 。 #2834
- 支持 UUID 在关键列。

改进：

- 当群集从配置文件中删除时，可以在不重新启动服务器的情况下删除群集。 #2777
- 从配置文件中删除外部字典时，可以在不重新启动服务器的情况下删除它们。 #2779

- 已添加 SETTINGS 支持 Kafka 表引擎。 Alexander Marshalov
- 改进的 UUID 数据类型（尚未完成）。 #2618
- 支持合并后的空部件 SummingMergeTree, CollapsingMergeTree 和 VersionedCollapsingMergeTree 引擎 #2815
- 已完成突变的旧记录将被删除 (ALTER DELETE). #2784
- 添加了 system.merge_tree_settings 桌子 基里尔*什瓦科夫
- 该 system.tables 表现在具有依赖列: dependencies_database 和 dependencies_table. 张冬
- 添加了 max_partition_size_to_drop 配置选项。 #2782
- 添加了 output_format_json_escape_forward_slashes 选项。 Alexander Bocharov
- 添加了 max_fetch_partition_retries_count 设置。 #2831
- 添加了 prefer_localhost_replica 用于禁用本地副本的首选项以及在不进程间交互的情况下转到本地副本的设置。 #2832
- 该 quantileExact 聚合函数返回 nan 在聚合在一个空的情况下 Float32 或 Float64 预备 李尚迪

错误修复:

- 删除了ODBC的连接字符串参数的不必要的转义，这使得无法建立连接。此错误发生在版本18.6.0中。
- 修正了处理逻辑 REPLACE PARTITION 复制队列中的命令。如果有两个 REPLACE 对于同一个分区的命令，不正确的逻辑可能会导致其中一个保留在复制队列中而无法执行。 #2814
- 修正了一个合并错误，当所有的数据部分都是空的（从合并或从形成的部分 ALTER DELETE 如果所有数据都被删除）。此错误出现在18.1.0版本。 #2930
- 修复了并发错误 Set 或 Join. 阿莫斯鸟
- 修正了 Block structure mismatch in UNION stream: different number of columns 发生的错误 UNION ALL 子查询内的查询，如果一个 SELECT 查询包含重复的列名。张冬
- 修复了连接到MySQL服务器时发生异常时的内存泄漏。
- 在查询错误的情况下修复了不正确的clickhouse客户端响应代码。
- 修正了包含DISTINCT的实例化视图的不正确行为。 #2795

向后不兼容的更改

- 删除了对分布式表的检查表查询的支持。

构建更改:

- 分配器已被替换: jemalloc 现在用来代替 tcmalloc. 在某些情况下，这增加了速度高达20%。但是，有些查询已经减慢了20%。在某些情况下，内存消耗减少了大约10%，稳定性得到了提高。由于竞争激烈的负载，用户空间和系统中的CPU使用率略有增加。 #2773
- 从子模块使用libressl。 #1983 #2807
- 从子模块使用unixodbc。 #2789
- 从子模块中使用mariadb-connector-c。 #2785
- 将功能性测试文件添加到存储库中，这些文件取决于测试数据的可用性（暂时不包含测试数据本身）。

ClickHouse释放18.6

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

- 增加了对ON表达式的支持，以便在语法上加入:
`JOIN ON Expr([table.]column ...) = Expr([table.]column, ...) [AND Expr([table.]column, ...) = Expr([table.]column, ...) ...]`
 表达式必须是由AND运算符连接的等式链。等式的每一侧都可以是其中一个表的列上的任意表达式。支持使用完全限定的列名 (table.name, database.table.name, table_alias.name, subquery_alias.name) 对于正确的表。 #2742
- 可以启用HTTPS进行复制。 #2760

改进:

- 服务器将其版本的补丁组件传递给客户端。有关修补程序版本组件的数据位于 system.processes 和 query_log.
#2646

ClickHouse释放18.5

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

- 添加了哈希函数 `murmurHash2_32` #2756.

改进:

- 现在你可以使用 `from_env` #2741 从环境变量设置配置文件中的值的属性。
- 增加了不区分大小写的版本 `coalesce`, `ifNull`, 和 `nullIf functions` #2752.

错误修复:

- 修复了启动副本时可能出现的错误 #2759.

ClickHouse释放18.4

碌莽禄,拢,010-68520682\<url>

新功能:

- 添加系统表: `formats`, `data_type_families`, `aggregate_function_combinators`, `table_functions`, `table_engines`, `collations` #2721.
- 增加了使用表函数代替表作为参数的能力 `remote` 或 `cluster table function` #2708.
- 支持 `HTTP Basic` 复制协议中的身份验证 #2727.
- 该 `has` 函数现在允许搜索数组中的数值 `Enum` 值 Maxim Khrisanfov.
- 支持添加任意消息分隔符从读取时 Kafka 阿莫斯鸟.

改进:

- 该 `ALTER TABLE t DELETE WHERE` 查询不会重写未受 `WHERE` 条件影响的数据部分 #2694.
- 该 `use_minimalistic_checksums_in_zookeeper` 备选案文 `ReplicatedMergeTree` 默认情况下启用表。此设置在版本 1.1.54378, 2018-04-16 中添加。不能再安装超过 1.1.54378 的版本。
- 支持运行 `KILL` 和 `OPTIMIZE` 指定的查询 `ON CLUSTER` 张冬.

错误修复:

- 修正了错误 `Column ... is not under an aggregate function and not in GROUP BY` 用于具有 IN 表达式的聚合。此错误出现在 18.1.0 版本。(bbdd780b)
- 修正了一个错误 `windowFunnel aggregate function` 张冬.
- 修正了一个错误 `anyHeavy` 聚合函数 (a2101df2)
- 使用时固定服务器崩溃 `countArray()` 聚合函数。

向后不兼容的更改:

- 参数 `Kafka` 发动机从改变 `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_schema, kafka_num_consumers])` 到 `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_row_delimiter, kafka_schema, kafka_num_consumers])`. 如果你的表使用 `kafka_schema` 或 `kafka_num_consumers` 参数, 你必须手动编辑元数据文件 `path/metadata/database/table.sql` 并添加 `kafka_row_delimiter` 参数 " 价值。

ClickHouse释放18.1

碌莽禄,拢,010-68520682\<url>

新功能:

- 支持 `ALTER TABLE t DELETE WHERE` 非复制 MergeTree 表的查询 (#2634).
- 支持任意类型的 `uniq*` 聚合函数族 (#2010).
- 支持比较运算符中的任意类型 (#2026).
- 该 `users.xml` 文件允许设置子网掩码的格式 `10.0.0.1/255.255.255.0`. 这对于在中间使用零的 IPv6 网络使用掩码是必要的 (#2637).
- 添加了 `arrayDistinct` 功能 (#2670).
- SummingMergeTree 引擎现在可以使用 `AggregateFunction` 类型列 (康斯坦丁*潘).

改进:

- 更改了发布版本的编号方案。现在第一部分包含发布年份（公元，莫斯科时区，减去2000），第二部分包含主要更改的数量（大多数版本的增加），第三部分是补丁版本。除非在更新日志中另有说明，否则版本仍然向后兼容。
- 更快地将浮点数转换为字符串（[阿莫斯鸟](#)）。
- 如果在插入过程中由于解析错误而跳过某些行（这可能与 `input_allow_errors_num` 和 `input_allow_errors_ratio` 启用设置），跳过的行数现在写入服务器日志（[列奥纳多*切奇](#)）。

错误修复：

- 修复了临时表的截断命令（[阿莫斯鸟](#)）。
- 修复了读取响应时出现网络错误时 ZooKeeper 客户端库中罕见的死锁（[c315200](#)）。
- 修复了转换为可空类型期间的错误（#1322）。
- 修正了不正确的结果 `maxIntersection()` 函数时间间隔的边界重合（[Michael Furmur](#)）。
- 修复了函数参数中 OR 表达式链的不正确转换（[晨兴-xc](#)）。
- 修复了包含 `IN (subquery)` 另一个子查询中的表达式（#2571）。
- 修复了分布式查询中使用不同版本的服务器之间的不兼容性 `CAST` 不是大写字母的函数（[fe8c4d6](#)）。
- 添加了对外部数据库管理系统查询的缺少标识符引用（#2635）。

向后不兼容的更改：

- 将包含数字零的字符串转换为 `DateTime` 不起作用。示例：`SELECT toDateTime('0')`。这也是原因 `DateTime DEFAULT '0'` 在表中不起作用，以及 `<null_value>0</null_value>` 在字典里解决方案：替换 `0` 与 `0000-00-00 00:00:00`。

ClickHouse 释放 1.1

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能：

- 添加了 `histogram` 聚合函数（[米哈伊尔*苏林](#)）。
- 现在 `OPTIMIZE TABLE ... FINAL` 可以在不指定分区的情况下使用 `ReplicatedMergeTree`（[阿莫斯鸟](#)）。

错误修复：

- 修复了在发送和下载复制数据时读取和写入套接字超时非常小的问题（一秒钟），这使得在网络或磁盘上存在负载时无法下载更大的部分（导致周期性尝试下载部分）。此错误发生在版本 1.1.54388。
- 修复了在 ZooKeeper 中使用 `chroot` 时在表中插入重复数据块的问题。
- 该 `has` 函数现在可以正常工作用于具有可为空元素的数组（#2115）。
- 该 `system.tables` 在分布式查询中使用表现在可以正常工作。该 `metadata_modification_time` 和 `engine_full` 列现在是非虚拟的。修复了仅从表中查询这些列时发生的错误。
- 固定如何空 `TinyLog` 表插入一个空数据块后工作（#2563）。
- 该 `system.zookeeper` 如果 ZooKeeper 中节点的值为 `NULL`，表就可以工作。

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能：

- 查询可以在发送 `multipart/form-data` 格式（在 `query` 字段），如果外部数据也被发送用于查询处理，这是有用的（[Olga Hvostikova](#)）。
- 增加了在读取 CSV 格式数据时启用或禁用处理单引号或双引号的功能。您可以在 `format_csv_allow_single_quotes` 和 `format_csv_allow_double_quotes` 设置（[阿莫斯鸟](#)）。
- 现在 `OPTIMIZE TABLE ... FINAL` 可以在不指定非复制变体的分区的情况下使用 `MergeTree`（[阿莫斯鸟](#)）。

改进：

- 在可以使用表索引时使用 IN 运算符提高性能、减少内存消耗并正确跟踪内存消耗（#2584）。
- 删除添加数据部分时校验和的冗余检查。当存在大量副本时，这一点很重要，因为在这些情况下，检查的总数等于 N^2 。
- 增加了对 `Array(Tuple(...))` 对于参数 `arrayEnumerateUniq` 功能（#2573）。
- 已添加 `Nullable` 支持 `runningDifference` 功能（#2594）。
- 当存在大量表达式时，改进了查询分析性能（#2572）。
- 更快地选择用于合并的数据部分 `ReplicatedMergeTree` 桌子更快地恢复 ZooKeeper 会话（#2597）。

- 该 `format_version.txt` 文件 `MergeTree` 如果表丢失，则重新创建表，如果在没有文件的情况下复制目录结构后启动 ClickHouse，这是有意义的 (Ciprian Hacman).

错误修复：

- 修复了与ZooKeeper一起工作时的错误，这可能会导致无法在重新启动服务器之前恢复表的会话和只读状态。
- 修复了与ZooKeeper一起工作时的错误，如果会话中断，可能会导致旧节点不被删除。
- 修正了一个错误 `quantileTDigest Float` 参数的函数（此错误在版本1.1.54388中引入）(米哈伊尔*苏林)。
- 修复了MergeTree表索引中的一个错误，如果主键列位于函数内部，用于在相同大小的有符号和无符号整数之间转换类型 (#2603)。
- 如果修复段错误 `macros` 使用，但它们不在配置文件中 (#2570)。
- 修复了重新连接客户端时切换到默认数据库的问题 (#2583)。
- 修正了当发生的错误 `use_index_for_in_with_subqueries` 设置被禁用。

安全修复：

- 当连接到MySQL时，发送文件不再可能 (`LOAD DATA LOCAL INFILE`)。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

- 支持 `ALTER TABLE t DELETE WHERE` 查询复制的表。添加了 `system.mutations` 表来跟踪这种类型的查询的进度。
- 支持 `ALTER TABLE t [REPLACE|ATTACH] PARTITION` 查询*MergeTree表。
- 支持 `TRUNCATE TABLE` 查询 (张冬)
- 几个新的 `SYSTEM` 复制表的查询 (`RESTART REPLICAS`, `SYNC REPLICA`, `[STOP|START] [MERGES|FETCHES|SENDS REPLICATED|REPLICATION QUEUES]`)。
- 增加了使用MySQL引擎和相应的表函数写入表的能力 (三弟)。
- 添加了 `url()` 表功能和 `URL` 表引擎 (Alexander Sapin)。
- 添加了 `windowFunnel` 聚合函数 (三弟)。
- 新 `startsWith` 和 `endsWith` 字符串的函数 (Vadim Plakhtinsky)。
- 该 `numbers()` 表函数现在允许您指定偏移量 (张冬)。
- 密码 `clickhouse-client` 可以交互输入。
- 服务器日志现在可以发送到系统日志 (Alexander Krasheninnikov)。
- 支持使用共享库源登录字典 (Alexander Sapin)。
- 支持自定义CSV分隔符 (伊万*朱可夫)
- 添加了 `date_time_input_format` 设置。如果将此设置切换到 '`best_effort`'，日期时间值将以各种格式读取。
- 添加了 `clickhouse-obfuscator` 用于数据混淆的实用程序。用法示例：发布性能测试中使用的数据。

实验特点:

- 增加了计算能力 `and` 只有在需要的地方才能参数 (阿纳斯塔西娅Tsarkova)
- Jit编译为本机代码现在可用于某些表达式 (pyos)。

错误修复：

- 对于具有以下内容的查询，不再显示重复项 `DISTINCT` 和 `ORDER BY`。
- 查询与 `ARRAY JOIN` 和 `arrayFilter` 不再返回不正确的结果。
- 修复了从嵌套结构读取数组列时的错误 (#2066)。
- 修复了使用HAVING子句分析查询时出现的错误，如 `HAVING tuple IN (...)`。
- 修复了使用递归别名分析查询时出现的错误。
- 修复了从REPLACINGMERGETREE读取过滤所有行的PREWHERE中的条件时出现的错误 (#2525)。
- 在HTTP界面中使用会话时，未应用用户配置文件设置。
- 修复了如何从clickhouse-local中的命令行参数应用设置。
- ZooKeeper客户端库现在使用从服务器接收的会话超时。
- 修正了ZooKeeper客户端库中的一个错误，当客户端等待服务器响应时间超过超时时间。
- 修剪部分的查询与分区键列的条件 (#2342)。
- 合并后，现在可以 `CLEAR COLUMN IN PARTITION` (#2315)。
- ODBC表函数中的类型映射已修复 (三弟)。
- 类型比较已修复 `DateTime` 有和没有时区 (Alexander Bocharov)。

- 修正了语法解析和格式化的 `CAST` 接线员。
- 固定插入到分布式表引擎的实例化视图中 ([Babacar Diassé](#))。
- 修正了从写入数据时的争用条件 `Kafka` 引擎到实例化视图 ([刘杨宽](#))。
- 固定 `ssrf` 中的 `remote()` 表函数。
- 固定退出行为 `clickhouse-client` 在多行模式下 ([#2510](#))。

改进:

- 复制表中的后台任务现在在线程池中执行，而不是在单独的线程中执行 ([Silviu Caragea](#))。
- 改进的 `LZ4` 压缩性能。
- 更快地分析具有大量联接和子查询的查询。
- 当有太多的网络错误时，`DNS` 缓存现在会自动更新。
- 如果由于其中一个实例化视图包含太多部件而无法插入表格插入，则不再发生表格插入。
- 纠正了事件计数器中的差异 `Query`, `SelectQuery`, 和 `InsertQuery`。
- 像这样的表达式 `tuple IN (SELECT tuple)` 如果元组类型匹配，则允许。
- 即使您没有配置 `ZooKeeper`，具有复制表的服务器也可以启动。
- 在计算可用 CPU 内核数时，现在考虑了 `cgroups` 的限制 ([Atri Sharma](#))。
- 在 `systemd` 配置文件中添加了配置目录的 `chown` ([米哈伊尔 Shiryaev](#))。

构建更改:

- `Gcc8` 编译器可用于构建。
- 增加了从子模块构建 `llvm` 的能力。
- `Librdkafka` 库的版本已更新为 `v0.11.4`。
- 增加了使用系统 `libcpuid` 库的能力。库版本已更新为 `0.4.0`。
- 使用 `vectorclass` 库修复了构建 ([Babacar Diassé](#))。
- `Cmake` 现在默认情况下为 `ninja` 生成文件 (如使用 `-G Ninja`)。
- 添加了使用 `libtinfo` 库而不是 `libtermcap` 的功能 ([Georgy Kondratiev](#))。
- 修复了 `Fedor Rawhide` 中的头文件冲突 ([#2520](#))。

向后不兼容的更改:

- 删除逃逸 `Vertical` 和 `Pretty*` 格式和删除 `VerticalRaw` 格式。
- 如果在分布式查询中同时使用版本 `1.1.54388` (或更高版本) 的服务器和版本较旧的服务器，并且查询具有 `cast(x, 'Type')` 表达式没有 `AS` 关键字并没有这个词 `cast` 以大写形式，将引发一个异常，并显示如下消息 `Not found column cast(0, 'UInt8') in block`. 解决方案：更新整个群集上的服务器。

碌莽祿, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682

错误修复:

- 修复了在某些情况下导致 `ZooKeeper` 操作阻塞的错误。

碌莽祿, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复:

- 修正了如果一个表有许多副本，复制队列的放缓。

碌莽祿, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

错误修复:

- 修复了 `ClickHouse` 与 `ZooKeeper` 服务器断开连接时，`ZooKeeper` 中的节点泄漏问题。

碌莽祿, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能:

- 增加了表功能 `file(path, format, structure)`. 从读取字节的示例 `/dev/urandom: In -s /dev/urandom /var/lib/clickhouse/user_files/random` `clickhouse-client -q "SELECT * FROM file('random', 'RowBinary', 'd UInt8') LIMIT 10".`

改进:

- 子查询可以包装在 () 括号以增强查询的可读性。例如: (SELECT 1) UNION ALL (SELECT 1).
- 简单 SELECT 从查询 system.processes 表不包括在 max_concurrent_queries 限制。

错误修复:

- 修正了不正确的行为 IN 从中选择时的运算符 MATERIALIZED VIEW.
- 修正了不正确的过滤分区索引的表达式, 如 partition_key_column IN (...).
- 固定无法执行 OPTIMIZE 在以下情况下对非领导副本进行查询 REANAME 在桌子上进行。
- 修复了执行时的授权错误 OPTIMIZE 或 ALTER 对非领导副本的查询。
- 固定的冻结 KILL QUERY.
- 修复了ZooKeeper客户端库中的错误, 这导致了手表丢失, 分布式的DDL队列冻结, 并在复制队列中的速度变慢, 如果非空 chroot 前缀在ZooKeeper配置中使用。

向后不兼容的更改:

- 删除对如下表达式的支持 (a, b) IN (SELECT (a, b)) (可以使用等效表达式 (a, b) IN (SELECT a, b)). 在以前的版本中, 这些表达式导致未确定 WHERE 过滤或导致的错误。

碌莽禄, 拢, 010-68520682\<url>戮卤箋拢, 010-68520682\<url>

新功能:

- 可以在不重新启动服务器的情况下更改日志记录级别。
- 添加了 SHOW CREATE DATABASE 查询。
- 该 query_id 可以传递给 clickhouse-client (肘部空间)。
- 新设置: max_network_bandwidth_for_all_users.
- 增加了对 ALTER TABLE ... PARTITION ... 为 MATERIALIZED VIEW.
- 在系统表中以未压缩形式添加有关数据部件大小的信息。
- 对分布式表的服务器到服务器加密支持 (<secure>1</secure> 在副本配置中 <remote_servers>).
- 表级别的配置 ReplicatedMergeTree 家庭, 以最大限度地减少存储在Zookeeper的数据量::

```
use_minimalistic_checksums_in_zookeeper = 1
```
- 的配置 clickhouse-client 提示。默认情况下, 服务器名称现在输出到提示符。可以更改服务器的显示名称。它也发送了 X-ClickHouse-Display-Name HTTP头 (基里尔Shvakov)。
- 多个逗号分隔 topics 可为指定 Kafka 发动机 (托比亚斯*亚当森)
- 当查询停止时 KILL QUERY 或 replace_running_query, 客户端接收 Query was canceled 异常而不是不完整的结果。

改进:

- ALTER TABLE ... DROP/DETACH PARTITION 查询在复制队列的前面运行。
- SELECT ... FINAL 和 OPTIMIZE ... FINAL 即使表具有单个数据部分, 也可以使用。
- A query_log 如果手动删除 (基里尔Shvakov), 则会在飞行中重新创建表格。
- 该 lengthUTF8 功能运行速度更快 (zhang2014)。
- 在同步刀片的性能提高 Distributed 表 (insert_distributed_sync = 1) 当有一个非常大的数量的碎片。
- 服务器接受 send_timeout 和 receive_timeout 从客户端设置并在连接到客户端时应用它们 (它们以相反的顺序应用: 服务器套接字的 send_timeout 被设置为 receive_timeout 值, 反之亦然)。
- 更强大的崩溃恢复异步插入 Distributed 桌子
- 的返回类型 countEqual 功能从更改 UInt32 到 UInt64 (谢磊).

错误修复:

- 修正了一个错误 IN 当表达式的左侧是 Nullable.
- 使用元组时, 现在返回正确的结果 IN 当某些元组组件位于表索引中时。
- 该 max_execution_time limit 现在可以正常使用分布式查询。
- 在计算复合列的大小时修正错误 system.columns 桌子
- 修复了创建临时表时的错误 CREATE TEMPORARY TABLE IF NOT EXISTS.
- 修正错误 StorageKafka (#2075)
- 修复了某些聚合函数的无效参数导致的服务器崩溃。
- 修正了防止错误 DETACH DATABASE 查询停止后台任务 ReplicatedMergeTree 桌子
- Too many parts 插入到聚合实例化视图时, 状态不太可能发生 (#2084)。
- 如果替换必须在同一级别上跟随另一个替换, 则更正了配置中替换的递归处理。

• 修正了创建元数据文件时的语法 `VIEW` 这使用一个查询 `UNION ALL`.

• `SummingMergeTree` 现在可以正常使用复合键对嵌套数据结构进行求和。

• 修复了在选择领导者时出现竞争条件的可能性 `ReplicatedMergeTree` 桌子

构建更改:

• 构建支持 `ninja` 而不是 `make` 和用途 `ninja` 默认情况下，构建版本。

• 重命名的软件包: `clickhouse-server-base` 在 `clickhouse-common-static`; `clickhouse-server-common` 在 `clickhouse-server`; `clickhouse-common-dbg` 在 `clickhouse-common-static-dbg`. 要安装，请使用 `clickhouse-server` `clickhouse-client`. 具有旧名称的软件包仍将加载到存储库中，以便向后兼容。

向后不兼容的更改:

- 如果在左侧指定了数组，则删除了IN表达式的特殊解释。以前，表达式 `arr IN (set)` 被解释为“at least one arr element belongs to the set”. 要在新版本中获得相同的行为，请编写 `arrayExists(x -> x IN (set), arr)`.
- 禁用套接字选项的不正确使用 `SO_REUSEPORT`，默认情况下，Poco库中未正确启用。请注意，在Linux上，不再有任何理由同时指定地址 `::` 和 `0.0.0.0` for listen – use just `::`，它允许监听通过IPv4和IPv6的连接（使用默认的内核配置设置）。您还可以通过指定以下命令恢复到以前版本中的行为 `<listen_reuse_port>1</listen_reuse_port>` 在配置。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

• 添加了 `system.macros` 更改配置文件时，宏的表和自动更新。

• 添加了 `SYSTEM RELOAD CONFIG` 查询。

• 添加了 `maxIntersections(left_col, right_col)` 聚合函数，它返回同时相交间隔的最大数目 `[left; right]`. 该 `maxIntersectionsPosition(left, right)` 函数返回的开始 “maximum” 间隔。**(Michael Furmur)**.

改进:

- 当在一个插入数据 `Replicated` 表，较少的请求是由 `ZooKeeper` （和大多数用户级错误已经从消失 `ZooKeeper` 日志）。
- 添加了为数据集创建别名的功能。示例: `WITH (1, 2, 3) AS set SELECT number IN set FROM system.numbers LIMIT 10`

错误修复:

• 修正了 `Illegal PREWHERE` 从合并表读取时出错 `Distributed` 桌子

• 添加了修复，允许您在仅支持IPv4的Docker容器中启动`clickhouse-server`。

• 修正了从系统读取时的争用条件 `system.parts_columns tables`.

• 同步插入到一个过程中删除双缓冲 `Distributed` 表，这可能导致连接超时。

• 修正了一个错误，导致过长的等待不可用的副本开始之前 `SELECT` 查询。

• 在固定不正确的日期 `system.parts` 桌子

• 修正了一个错误，使得它无法在插入数据 `Replicated` 表if `chroot` 是非空的配置 `ZooKeeper` 集群。

• 修正了一个空的垂直合并算法 `ORDER BY` 桌子

• 恢复了在对远程表的查询中使用字典的能力，即使这些字典不存在于请求者服务器上。此功能在版本1.1.54362中丢失。

• 恢复查询的行为，如 `SELECT * FROM remote('server2', default.table) WHERE col IN (SELECT col2 FROM default.table)` 当右侧的 `IN` 应该使用远程 `default.table` 而不是当地的 此行为在版本1.1.54358中被破坏。

• 删除了无关的错误级别日志记录 `Not found column ... in block.`

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

• 聚合不 `GROUP BY` 对于一个空集（如 `SELECT count(*) FROM table WHERE 0`）现在返回一个结果，其中一行为聚合函数带有`null`值，符合SQL标准。要恢复旧行为（返回一个空结果），请设置

`empty_result_for_aggregation_by_empty_set` 到1。

• 增加了类型转换 `UNION ALL`. 不同的别名被允许 `SELECT` 在职位 `UNION ALL`，符合SQL标准。

• 任意表达式支持 `LIMIT BY` 条款 以前，只能使用以下内容产生的列 `SELECT`.

• 的索引 `MergeTree` 表用于以下情况 `IN` 应用于来自主键列的表达式元组。示例: `WHERE (UserID, EventDate) IN ((123, '2000-01-01'), ...)` (Anastasiya Tsarkova)

- 添加了 `clickhouse-copier` 用于在群集之间复制和重新分布数据的工具（测试版）。
- 添加了一致的哈希函数: `yandexConsistentHash`, `jumpConsistentHash`, `sumburConsistentHash`. 它们可以用作分片密钥，以减少后续重新分片期间的网络流量。
- 新增功能: `arrayAny`, `arrayAll`, `hasAny`, `hasAll`, `arrayIntersect`, `arrayResize`.
- 添加了 `arrayCumSum` 功能（哈维桑塔纳）。
- 添加了 `parseDateTimeBestEffort`, `parseDateTimeBestEffortOrZero`，和 `parseDateTimeBestEffortOrNull` 用于从包含各种可能格式的文本的字符串中读取`DateTime`的函数。
- 数据可以在更新期间从外部字典部分重新加载（加载只是记录，其中指定字段的值大于先前的下载）（Arsen Hakobyan）。
- 添加了 `cluster` 表功能。示例: `cluster(cluster_name, db, table)`. 该 `remote` 表函数可以接受集群名称作为第一个参数，如果它被指定为标识符。
- 该 `remote` 和 `cluster` 表函数可用于 `INSERT` 查询。
- 添加了 `create_table_query` 和 `engine_full` 虚拟列到 `system.tables` 桌子 该 `metadata_modification_time` 列是虚拟的。
- 添加了 `data_path` 和 `metadata_path` 列 `system.tables` 和 `system.databases` 表，并添加了 `path` 列到 `system.parts` 和 `system.parts_columns` 桌子
- 添加了关于合并的其他信息 `system.part_log` 桌子
- 一个任意的分区键可以用于 `system.query_log` 表（基里尔Shvakov）。
- 该 `SHOW TABLES` 查询现在还显示临时表。添加临时表和 `is_temporary` 列到 `system.tables` （张2014）。
- 已添加 `DROP TEMPORARY TABLE` 和 `EXISTS TEMPORARY TABLE` 查询（zhang2014）。
- 支持 `SHOW CREATE TABLE` 对于临时表（zhang2014）。
- 添加了 `system_profile` 内部进程使用的设置的配置参数。
- 支持加载 `object_id` 作为一个属性 MongoDB 字典（帕维尔*利特维年科）。
- 阅读 `null` 作为加载数据的外部字典与时的默认值 MongoDB 资料来源（帕维尔*利特维年科）。
- 阅读 `DateTime` 在值 `Values` 从不带单引号的Unix时间戳格式化。
- 故障转移支持 `remote` 当某些副本缺少请求的表时，表函数。
- 运行时可以在命令行中复盖配置设置 `clickhouse-server`. 示例: `clickhouse-server --logger.level=information`.
- 实施了 `empty` 从功能 `FixedSize` 参数：如果字符串完全由空字节组成，则函数返回1(zhang2014)。
- 添加了 `listen_try` 如果某些地址无法侦听，则在不退出的情况下侦听至少一个侦听地址的配置参数（对于禁用IPv4或IPv6支持的系统非常有用）。
- 添加了 `VersionedCollapsingMergeTree` 表引擎。
- 对于行和任意数字类型的 `library` 字典源。
- `MergeTree` 表可以在没有主键的情况下使用（您需要指定 `ORDER BY tuple()`).
- A `Nullable` 类型可以是 `CAST` 到非-`Nullable` 如果参数不是，则键入 `NUL`.
- `RENAME TABLE` 可以进行 `VIEW`.
- 添加了 `throwIf` 功能。
- 添加了 `odbc_default_field_size` 选项，它允许您扩展从ODBC源加载的值的最大大小（默认情况下为1024）。
- 该 `system.processes` 表和 `SHOW PROCESSLIST` 现在有 `is_cancelled` 和 `peak_memory_usage` 列。

改进:

- 结果的限制和配额不再应用于以下内容的中间数据 `INSERT SELECT` 查询或 `SELECT` 子查询。
- 更少的虚假触发 `force_restore_data` 当检查的状态 `Replicated` 服务器启动时的表。
- 添加了 `allow_distributed_ddl` 选项。
- 表达式中不允许使用非确定性函数 `MergeTree` 表键。
- 从替换文件 `config.d` 目录按字母顺序加载。
- 的改进的性能 `arrayElement` 函数在常量多维数组的情况下，以空数组作为元素之一。示例: `[[1], []][x]`.
- 当使用具有非常大的替换（例如，非常大的IP网络列表）的配置文件时，服务器现在启动速度更快。
- 运行查询时，表值函数运行一次。前情提要，`remote` 和 `mysql` 表值函数执行两次相同的查询以从远程服务器检索表结构。
- 该 `MkDocs` 使用文档生成器。
- 当您尝试删除表列时 `DEFAULT/MATERIALIZED` 取决于其他列的表达式，会抛出异常（zhang2014）。
- 增加了解析文本格式的空行作为数字0的能力 `Float` 数据类型。此功能以前可用，但在版本1.1.54342中丢失。
- `Enum` 值可以用于 `min`, `max`, `sum` 和其他一些功能。在这些情况下，它使用相应的数值。此功能以前可用，但在版本1.1.54337中丢失。
- 已添加 `max_expanded_ast_elements` 递归扩展别名后限制AST的大小。

错误修复:

- 修复了错误地从子查询中删除不必要的列或未从包含以下内容的子查询中删除必要列的情况 UNION ALL.
- 修正了合并的错误 ReplacingMergeTree 桌子
- 在固定的同步插入 Distributed 表 (insert_distributed_sync = 1).
- 固定段错误的某些用途 FULL 和 RIGHT JOIN 在子查询中使用重复的列。
- 固定段错误的某些用途 replace_running_query 和 KILL QUERY.
- 固定的顺序 source 和 last_exception 在列 system.dictionaries 桌子
- 修正了一个错误，当 DROP DATABASE 查询没有删除带有元数据的文件。
- 修正了 DROP DATABASE 查询为 Dictionary 数据库。
- 固定的低精度 uniqHLL12 和 uniqCombined 功能基数大于100万个项目 (Alex克斯Bocharov)。
- 修复了在必要时计算隐式默认值，以便同时计算默认显式表达式 INSERT 查询 (zhang2014)。
- 修正了一个罕见的情况下，当一个查询 MergeTree 表不能完成 (陈星-xc)。
- 修正了运行时发生的崩溃 CHECK 查询为 Distributed 如果所有分片都是本地的 (chenxing.xc)。
- 修复了使用正则表达式的函数的轻微性能回归。
- 修复了从复杂表达式创建多维数组时的性能回归。
- 修正了一个错误，可能会导致一个额外的 FORMAT 部分出现在一个 .sql 具有元数据的文件。
- 修复了导致 max_table_size_to_drop 尝试删除时应用的限制 MATERIALIZED VIEW 查看显式指定的表。
- 修复了与旧客户端的不兼容性 (旧客户端有时会发送数据 DateTime('timezone') 类型，他们不明白)。
- 修复了阅读时的错误 Nested 使用以下方式添加的结构的列元素 ALTER 但是，这是空的旧分区，当这些列的条件移动到 PREWHERE.
- 修正了通过虚拟过滤表时的错误 _table 查询中的列 Merge 桌子
- 修复了使用时的错误 ALIAS 列 Distributed 桌子
- 修正了一个错误，使得动态编译不可能从聚合函数的查询 quantile 家人
- 修复了查询执行管道中极少数情况下使用时发生的争用条件 Merge 具有大量表的表，并且当使用 GLOBAL 子查询。
- 修复了将不同大小的数组传递给 arrayReduce 使用来自多个参数的聚合函数时的函数。
- 禁止使用与查询 UNION ALL 在一个 MATERIALIZED VIEW.
- 修正了初始化过程中的错误 part_log 服务器启动时的系统表 (默认情况下, part_log 被禁用)。

向后不兼容的更改:

- 删除了 distributed_ddl_allow_replicated_alter 选项。默认情况下启用此行为。
- 删除了 strict_insert_defaults 设置。如果您使用此功能，请写入 clickhouse-feedback@yandex-team.com.
- 删除了 UnsortedMergeTree 引擎

碌莽禄,拢,010-68520682\<url>戮漏鹿芦,酶,虏卤赂拢,110102005602

- 在分布式DDL查询和分布式表的构造函数中添加了用于定义集群名称的宏支持: CREATE TABLE distr ON CLUSTER '{cluster}' (...) ENGINE = Distributed('{cluster}', 'db', 'table').
- 现在像查询 SELECT ... FROM table WHERE expr IN (subquery) 使用处理 table 指数。
- 在插入到复制表时改进了重复项的处理，因此它们不再减慢复制队列的执行速度。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682

此版本包含以前版本1.1.54337的错误修复:

- 修正了1.1.54337中的回归：如果默认用户具有只读访问权限，则服务器拒绝启动消息 Cannot create database in readonly mode.
- 修正了1.1.54337中的回归：在具有systemd的系统上，无论配置如何，日志总是写入syslog;看门狗脚本仍然使用 init.d。
- 修正了1.1.54337中的回归：Docker映像中错误的默认配置。
- 修正GraphiteMergeTree的非确定性行为（您可以在日志消息中看到它 Data after merge is not byte-identical to the data on another replicas）。
- 修复了优化查询到复制表后可能导致合并不一致的错误（您可能会在日志消息中看到它 Part ... intersects the previous part）。
- 当目标表中存在具体化列时，缓冲区表现在可以正常工作（by zhang2014）。
- 修复了实现NULL的错误。

新功能:

- 增加了对多维数组和元组存储的支持 (`Tuple` 表中的数据类型)。
- 支持表函数 `DESCRIBE` 和 `INSERT` 查询。增加了对子查询的支持 `DESCRIBE`。例: `DESC TABLE remote('host', default.hits); DESC TABLE (SELECT 1); INSERT INTO TABLE FUNCTION remote('host', default.hits)`。支持 `INSERT INTO TABLE` 除了 `INSERT INTO`。
- 改进了对时区的支持。该 `DateTime` 数据类型可以使用用于以文本格式进行分析和格式化的时区进行注释。示例: `DateTime('Europe/Moscow')`。当在函数中指定时区时 `DateTime` 参数, 返回类型将跟踪时区, 并且值将按预期显示。
- 新增功能 `toTimeZone`, `timeDiff`, `toQuarter`, `toRelativeQuarterNum`。该 `toRelativeHour/Minute/Second` 函数可以采用类型的值 `Date` 作为参数。该 `now` 函数名称区分大小写。
- 添加了 `toStartOfFifteenMinutes` 功能 (基里尔 Shvakov)。
- 添加了 `clickhouse format` 用于格式化查询的工具。
- 添加了 `format_schema_path` configuration parameter (Marek Vavruša)。It is used for specifying a schema in Cap'n Proto 格式。架构文件只能位于指定的目录中。
- 增加了对配置替换的支持 (`incl` 和 `conf.d`) 外部字典和模型的配置 (帕维尔*亚库宁)。
- 添加了一列文档 `system.settings` 表 (基里尔 Shvakov)。
- 添加了 `system.parts_columns` 表中的每个数据部分的列大小信息 `MergeTree` 桌子
- 添加了 `system.models` 包含已加载信息的表 `CatBoost` 机器学习模型。
- 添加了 `mysql` 和 `odbc` 表函数和对应 MySQL 和 ODBC 用于访问远程数据库的表引擎。此功能处于测试阶段。
- 增加了传递类型参数的可能性 `AggregateFunction` 为 `groupArray` 聚合函数 (这样你就可以创建一些聚合函数的状态数组)。
- 删除了对聚合函数组合器的各种组合的限制。例如, 您可以使用 `avgForEachIf` 以及 `avgIfForEach` 聚合函数, 它们具有不同的行为。
- 该 `-ForEach` 聚合函数 `combinator` 是针对多个参数的聚合函数的情况进行扩展的。
- 增加了对聚合函数的支持 `Nullable` 即使是函数返回非参数的情况-`Nullable` 结果 (添加 Silviu Caragea 的贡献)。示例: `groupArray`, `groupUniqArray`, `topK`。
- 添加了 `max_client_network_bandwidth` 为 `clickhouse-client` (基里尔*什瓦科夫)。
- 用户与 `readonly = 2` setting are allowed to work with TEMPORARY tables (CREATE, DROP, INSERT...) (Kirill Shvakov).
- 增加了对使用多个消费者的 support `Kafka` 引擎 扩展的配置选项 `Kafka` (Marek Vavruša).
- 添加了 `intExp3` 和 `intExp4` 功能。
- 添加了 `sumKahan` 聚合函数。
- 添加了 `to*Number*OrNull` 函数, 其中 `*Number*` 是数字类型。
- 增加了对 `WITH a` 的子句 `INSERT SELECT` 查询 (作者: zhang2014)。
- 添加设置: `http_connection_timeout`, `http_send_timeout`, `http_receive_timeout`。特别是, 这些设置用于下载用于复制的数据部分。如果网络过载, 更改这些设置可以更快地进行故障转移。
- 增加了对 `ALTER` 对于类型的表 `Null` (Anastasiya Tsarkova)
- 该 `reinterpretAsString` 函数扩展为连续存储在内存中的所有数据类型。
- 添加了 `--silent` 选项的 `clickhouse-local` 工具 它禁止在 `stderr` 中打印查询执行信息。
- 增加了对读取类型值的支持 `Date` 从使用单个数字而不是两个数字 (Amos Bird) 指定月份和/或月份日的格式的文本。

性能优化:

- 改进聚合函数的性能 `min`, `max`, `any`, `anyLast`, `anyHeavy`, `argMin`, `argMax` 从字符串参数。
- 改进功能的性能 `isInfinite`, `isFinite`, `isNaN`, `roundToExp2`。
- 改进了解析和格式化的性能 `Date` 和 `DateTime` 以文本格式键入值。
- 改进了解析浮点数的性能和精度。
- 降低内存使用量 `JOIN` 在左部分和右部分具有不包含在相同名称的列的情况下 `USING`。
- 改进聚合函数的性能 `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr` 通过降低计算稳定性。旧函数的名称下可用 `varSampStable`, `varPopStable`, `stddevSampStable`, `stddevPopStable`, `covarSampStable`, `covarPopStable`, `corrStable`。

错误修复:

- 固定数据重复数据删除运行后 `DROP` 或 `DETACH PARTITION` 查询。在以前的版本中, 删除分区并再次插入相同的数据不起作用, 因为插入的块被认为是重复的。

- 修复了可能导致错误解释的错误 WHERE 条款 CREATE MATERIALIZED VIEW 查询与 POPULATE .
- 修正了在使用 root_path 在参数 zookeeper_servers 配置。
- 通过固定意外的结果 Date 论据 toStartOfDay .
- 修正了 addMonths 和 subtractMonths 函数和算术 INTERVAL n MONTH 在情况下，当结果有前一年。
- 增加了缺少的支持 UUID 数据类型 DISTINCT , JOIN , 和 uniq 聚合函数和外部字典 (叶夫根尼伊万诺夫) 。 支持 UUID 仍然是不完整的。
- 固定 SummingMergeTree 行为的情况下，当行相加为零。
- 各种修复 Kafka engine (Marek Vavruša).
- 修正了不正确的行为 Join 表引擎 (阿莫斯鸟) 。
- 修复了 FreeBSD 和 OS X 下不正确的分配器行为。
- 该 extractAll 函数现在支持空匹配。
- 修复了阻止使用的错误 libressl 而不是 openssl .
- 修正了 CREATE TABLE AS SELECT 从临时表查询。
- 修复了更新复制队列的非原子性。这可能导致副本在服务器重新启动之前不同步。
- 修正了可能的溢出 gcd , lcm 和 modulo (% 运营商) (Maks Skorokhod) 。
- preprocessed 现在更改后创建文件 umask (umask 可以在配置中更改) 。
- 修正了部分的背景检查中的错误 (MergeTreePartChecker) 使用自定义分区密钥时。
- 元组的固定解析 (的值 Tuple 数据类型) 的文本格式。
- 改进了有关传递到的不兼容类型的错误消息 multilf , array 和其他一些功能。
- 重新设计的支持 Nullable 类型。修复了可能导致服务器崩溃的错误。修正了与几乎所有其他错误 NULL 支持 : insert SELECT 中的类型转换不正确，HAVING 和 PREWHERE 中对 Nullable 的支持不足，join_use_nulls 模式，可以为 Null 的类型作为参数 OR 操作员等。
- 修正了与数据类型的内部语义相关的各种错误。例子：不必要的总结 Enum 输入字段 SummingMergeTree ; 对齐 Enum 类型 Pretty 格式等。
- 对复合列的允许组合进行更严格的检查。
- 修复了指定一个非常大的参数时的溢出 FixedString 数据类型。
- 修正了一个错误 topK 一般情况下的聚合函数。
- 在聚合函数的n元变体的参数中添加了对数组大小相等性的缺失检查。 -Array combinator
- 修正了一个错误 --pager 为 clickhouse-client (作者 : ks1322) 。
- 固定的精度 exp10 功能。
- 固定的行为 visitParamExtract 功能更好地符合文档。
- 修复了指定不正确的数据类型时的崩溃。
- 固定的行为 DISTINCT 在所有列都是常量的情况下。
- 在使用的情况下固定的查询格式 tupleElement 使用复数常量表达式作为元组元素索引的函数。
- 修正了一个错误 Dictionary 表 range_hashed 字典
- 修正了导致结果中的过多行的错误 FULL 和 RIGHT JOIN (阿莫斯鸟) 。
- 修复了在创建和删除临时文件时的服务器崩溃 config.d 配置重新加载期间的目录。
- 修正了 SYSTEM DROP DNS CACHE 查询：缓存已刷新，但群集节点的地址未更新。
- 固定的行为 MATERIALIZED VIEW 执行后 DETACH TABLE for the table under the view (Marek Vavruša) .

构建改进:

- 该 pbuilder 工具用于构建。构建过程几乎完全独立于构建主机环境。
- 单个构建用于不同的操作系统版本。软件包和二进制文件已经与各种 Linux 系统兼容。
- 添加了 clickhouse-test 包。它可用于运行功能测试。
- 现在可以将源代码包发布到存储库。它可以用来在不使用 GitHub 的情况下重现构建。
- 增加了有限的集成与特拉维斯 CI 。由于 Travis 中的构建时间限制，仅测试调试构建并运行有限的测试子集。
- 增加了对 Cap'n'Proto 在默认构建中。
- 更改文档来源的格式 Restricted Text 到 Markdown.
- 增加了对 systemd (弗拉基米尔*斯米尔诺夫) 。默认情况下，由于与某些操作系统映像不兼容，它被禁用，并且可以手动启用。
- 用于动态代码生成， clang 和 lld 嵌入到 clickhouse 二进制 它们也可以被调用为 clickhouse clang 和 clickhouse lld .
- 从代码中删除 GNU 扩展的使用。启用 -Wextra 选项。当与建设 clang 默认值为 libc++ 而不是 libstdc++ .
- 提取 clickhouse_parsers 和 clickhouse_common_io 库，以加快各种工具的构建。

向后不兼容的更改:

- 标记的格式 `Log` 键入包含以下内容的表 `Nullable` 列以向后不兼容的方式进行了更改。如果你有这些表，你应该将它们转换为 `TinyLog` 在启动新服务器版本之前键入。要做到这一点，替换 `ENGINE = Log` 与 `ENGINE = TinyLog` 在相应的 `.sql` 文件中的 `metadata` 目录。如果你的桌子没有 `Nullable` 列或表的类型不是 `Log`，那么你什么都不需要做。
- 删除了 `experimental_allow_extended_storage_definition_syntax` 设置。现在，此功能默认启用。
- 该 `runningIncome` 函数重命名为 `runningDifferenceStartingWithFirstValue` 为了避免混淆。
- 删除了 `FROM ARRAY JOIN arr` 在 `FROM with no table(Amos Bird)` 之后直接指定数组连接时的语法。
- 删除了 `BlockTabSeparated` 仅用于演示目的的格式。
- 更改聚合函数的状态格式 `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. 如果您已将这些聚合函数的状态存储在表中（使用 `AggregateFunction` 数据类型或具体化视图与相应状态），请写信给 clickhouse-feedback@yandex-team.com。
- 在以前的服务器版本中，有一个未记录的功能：如果聚合函数依赖于参数，则仍然可以在 `AggregateFunction` 数据类型中指定它而不带参数。示例：`AggregateFunction(quantiles, UInt64)` 而不是 `AggregateFunction(quantiles(0.5, 0.9), UInt64)`. 此功能已丢失。虽然它没有记录，但我们计划在未来的版本中再次支持它。
- 枚举数据类型不能用于最小/最大聚合函数。这种能力将在下一个版本中返回。

升级时请注意：

- 当在群集上执行滚动更新时，当某些副本运行旧版本的 ClickHouse，而某些副本运行新版本时，复制会暂时停止，并且消息 `unknown parameter 'shard'` 出现在日志中。更新集群的所有副本后，复制将继续。
- 如果群集服务器上运行不同版本的 ClickHouse，则使用以下函数的分布式查询可能会产生不正确的结果：`varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. 您应该更新所有群集节点。

更新日志2017

ClickHouse释放1.1.54327,2017-12-21

此版本包含以前版本1.1.54318的错误修复：

- 修复了可能导致数据丢失的复制中可能存在的争用条件的错误。此问题影响版本1.1.54310和1.1.54318。如果将其中一个版本用于复制的表，则强烈建议进行更新。此问题显示在日志中的警告消息，如 `Part ... from own log doesn't exist.` 即使您在日志中没有看到这些消息，问题也是相关的。

碌莽禄,拢,010-68520682\<url>

此版本包含以前版本1.1.54310的错误修复：

- 修复了 SummingMergeTree 引擎中合并过程中错误的行删除
- 修复了未复制的 MergeTree 引擎中的内存泄漏
- 修复了 MergeTree 引擎中频繁插入的性能下降
- 修复了导致复制队列停止运行的问题
- 固定服务器日志的轮换和归档

ClickHouse释放1.1.54310,2017-11-01

新功能：

- MergeTree 表引擎系列的自定义分区键。
- **卡夫卡** 表引擎。
- 增加了对加载的支持 `CatBoost` 模型并将其应用到 ClickHouse 中存储的数据。
- 增加了对 UTC 非整数偏移的时区的支持。
- 增加了对具有时间间隔的算术运算的支持。
- 日期和日期时间类型的值范围扩展到 2105 年。
- 添加了 `CREATE MATERIALIZED VIEW x TO y` 查询（指定用于存储实例化视图数据的现有表）。
- 添加了 `ATTACH TABLE` 不带参数的查询。
- 将 SummingMergeTree 表中名称以 -Map 结尾的嵌套列的处理逻辑提取到 `sumMap` 聚合函数中。现在，您可以显式指定此类列。
- IP trie 字典的最大大小增加到 128M 条目。

- 添加了 `getSizeOfEnumType` 函数。
- 添加了 `sumWithOverflow` 聚合函数。
- 增加了对 Cap'n Proto 输入格式的支持。
- 使用 `zstd` 算法时，您现在可以自定义压缩级别。

向后不兼容的更改:

- 不允许使用内存以外的引擎创建临时表。
- 不允许使用 `View` 或 `MaterializedView` 引擎显式创建表。
- 在创建表期间，新检查将验证采样键表达式是否包含在主键中。

错误修复:

- 修复了同步插入到分布式表中时的挂断问题。
- 修复了复制表中部分的非原子添加和删除。
- 插入到实例化视图中的数据不会遭受不必要的重复数据删除。
- 对本地副本滞后且远程副本不可用的分布式表执行查询不会再导致错误。
- 用户不需要访问权限 `default` 数据库创建临时表了。
- 修复了在指定数组类型时不带参数的崩溃。
- 修复了包含服务器日志的磁盘卷已满时的挂机问题。
- 修复了 `unix` 时代的第 1 周 `toRelativeWeekNum` 函数的溢出。

构建改进:

- 几个第三方库（特别是 Poco）被更新并转换为 `git` 子模块。

ClickHouse 释放 1.1.54304, 2017-10-19

新功能:

- 本机协议中的 TLS 支持（要启用，请设置 `tcp_ssl_port` 在 `config.xml`）。

错误修复:

- `ALTER` 对于复制的表现在尝试尽快开始运行。
- 使用设置读取数据时修复崩溃 `preferred_block_size_bytes=0`。
- 固定的崩溃 `clickhouse-client` 按下时 `Page Down`
- 正确解释某些复杂的查询 `GLOBAL IN` 和 `UNION ALL`
- `FREEZE PARTITION` 现在总是以原子方式工作。
- 空 POST 请求现在返回代码 411 的响应。
- 修正了像表达式的解释错误 `CAST(1 AS Nullable(UInt8))`。
- 修正了读取时的错误 `Array(Nullable(String))` 从列 `MergeTree` 桌子
- 修正了解析查询时崩溃，如 `SELECT dummy AS dummy, dummy AS b`
- 用户正确更新无效 `users.xml`
- 可执行字典返回非零响应代码时的正确处理。

ClickHouse 释放 1.1.54292, 2017-09-20

新功能:

- 添加了 `pointInPolygon` 用于处理坐标平面上的坐标的函数。
- 添加了 `sumMap` 用于计算数组总和的聚合函数，类似于 `SummingMergeTree`。
- 添加了 `trunc` 功能。改进舍入函数的性能 (`round`, `floor`, `ceil`, `roundToExp2`) 并 `corrected` 正了他们如何工作的逻辑。改变的逻辑 `roundToExp2` 分数和负数的功能。
- ClickHouse 可执行文件现在对 libc 版本的依赖性较低。同样的 ClickHouse 可执行文件可以在各种各样的 Linux 系统上运行。使用编译的查询（使用设置）时仍然存在依赖关系 `compile = 1`，默认情况下不使用）。
- 减少了动态编译查询所需的时间。

错误修复:

- 修正了有时产生的错误 `part ... intersects previous part` 消息和副本的一致性减弱。
- 修正了一个错误，导致服务器锁定，如果 ZooKeeper 在关闭过程中不可用。
- 恢复副本时删除了过多的日志记录。

- 修复了UNION ALL实现中的错误。
- 修复了在块中的第一列具有数组类型时发生的concat函数中的错误。
- 进度现在在系统中正确显示。合并表。

ClickHouse释放1.1.54289,2017-09-13

新功能:

- `SYSTEM` 服务器管理查询: `SYSTEM RELOAD DICTIONARY`, `SYSTEM RELOAD DICTIONARIES`, `SYSTEM DROP DNS CACHE`, `SYSTEM SHUTDOWN`, `SYSTEM KILL`.
- 添加了用于处理数组的函数: `concat`, `arraySlice`, `arrayPushBack`, `arrayPushFront`, `arrayPopBack`, `arrayPopFront`.
- 已添加 `root` 和 `identity` ZooKeeper配置的参数。这允许您隔离同一个ZooKeeper集群上的单个用户。
- 添加聚合函数 `groupBitAnd`, `groupBitOr`, 和 `groupBitXor` (为了兼容, 它们也可以在名称下使用 `BIT_AND`, `BIT_OR`, 和 `BIT_XOR`).
- 通过在文件系统中指定套接字, 可以从MySQL加载外部字典。
- 外部字典可以通过SSL从MySQL加载 (`ssl_cert`, `ssl_key`, `ssl_ca` 参数)。
- 添加了 `max_network_bandwidth_for_user` 设置为限制每个用户查询的总带宽使用。
- 支持 `DROP TABLE` 对于临时表。
- 支持阅读 `DateTime` 从Unix时间戳格式的值 `CSV` 和 `JSONEachRow` 格式。
- 分布式查询中的滞后副本现在默认排除 (默认阈值为5分钟)。
- 在ALTER期间使用FIFO锁定: 对于连续运行的查询, ALTER查询不会无限期地阻止。
- 选项设置 `umask` 在配置文件中。
- 改进了查询的性能 `DISTINCT`.

错误修复:

- 改进了在ZooKeeper中删除旧节点的过程。以前, 如果插入非常频繁, 旧节点有时不会被删除, 这导致服务器关闭速度缓慢等等。
- 修正了选择主机连接到ZooKeeper时的随机化。
- 修复了在分布式查询中排除滞后副本, 如果副本是localhost。
- 修正了一个错误, 其中在一个数据部分 `ReplicatedMergeTree` 运行后表可能会被打破 `ALTER MODIFY` 在一个元素 `Nested` 结构。
- 修复了可能导致SELECT查询执行以下操作的错误 “hang”.
- 对分布式DDL查询的改进。
- 修正了查询 `CREATE TABLE ... AS <materialized view>`.
- 解决了在僵局 `ALTER ... CLEAR COLUMN IN PARTITION` 查询为 `Buffer` 桌子
- 修正了无效的默认值 `Enum s (0, 而不是最小)` 使用时 `JSONEachRow` 和 `TSKV` 格式。
- 解决了使用字典时僵尸进程的外观 `executable` 资料来源。
- 修正了HEAD查询的段错误。

改进开发和组装ClickHouse的工作流程:

- 您可以使用 `pbuilder` 建造克里克豪斯
- 您可以使用 `libc++` 而不是 `libstdc++` 对于构建在Linux上。
- 添加了使用静态代码分析工具的说明: `Coverage`, `clang-tidy`, `cppcheck`.

升级时请注意:

- `MergeTree`设置现在有一个更高的默认值 `max_bytes_to_merge_at_max_space_in_pool` (要合并的数据部分的最大总大小, 以字节为单位) : 它已从100GiB增加到150GiB。这可能会导致服务器升级后运行大型合并, 这可能会导致磁盘子系统的负载增加。如果服务器上的可用空间小于正在运行的合并总量的两倍, 这将导致所有其他合并停止运行, 包括小数据部分的合并。因此, 插入查询将失败, 并显示消息 “Merges are processing significantly slower than inserts.” 使用 `SELECT * FROM system.merges` 查询监控情况。您还可以检查 `DiskSpaceReservedForMerge` 度量在 `system.metrics` 表, 或石墨。你不需要做任何事情来解决这个问题, 因为一旦大合并完成, 问题就会自行解决。如果您发现这是不可接受的, 则可以恢复以前的值 `max_bytes_to_merge_at_max_space_in_pool` 设置。要做到这一点, 请转到 `<merge_tree>` 在配置部分。`.xml`, 设置
`<merge_tree>`<max_bytes_to_merge_at_max_space_in_pool>107374182400</max_bytes_to_merge_at_max_space_in_pool>` 并重新启动服务器。

碌莽禄,拢,010-68520682\<url>

- 这是一个错误修正版本，以前的1.1.54282版本。它修复了ZooKeeper中部件目录中的泄漏。

碌莽禄,拢,010-68520682\<url>

此版本包含以前版本1.1.54276的错误修复：

- 固定 DB::Exception: Assertion violation: !_path.empty() 当插入到分布式表中。
- 如果输入数据以";"开头，则以RowBinary格式插入时修复了解析。
- Errors during runtime compilation of certain aggregate functions (e.g. groupArray()).

碌莽禄,拢,010-68520682\<url>

新功能：

- 为选择查询添加了一个可选的WITH部分。查询示例：WITH 1+1 AS a SELECT a, a*a
- INSERT可以在分布式表中同步执行：仅在所有分片上保存所有数据后才返回OK。这是由设置 insert_distributed_sync=1激活的。
- 添加了用于处理16字节标识符的UUID数据类型。
- 添加了CHAR, FLOAT和其他类型的别名，以便与Tableau兼容。
- 添加了toyyyymm, toYYYYMMDD和toyyyyyymmddhhmmss将时间转换为数字的功能。
- 您可以使用IP地址（与主机名一起使用）来标识群集DDL查询的服务器。
- 增加了对函数中非常量参数和负偏移的支持 substring(str, pos, len)。
- 添加了max_size参数 groupArray(max_size)(column) 聚合函数，并优化了其性能。

主要变化：

- 安全性改进：所有服务器文件都使用0640权限创建（可以通过更改 <umask> 配置参数）。
- 改进了语法无效的查询的错误消息。
- 在合并mergetree大部分数据时，显着降低了内存消耗并提高了性能。
- 显着提高了ReplacingMergeTree引擎的数据合并性能。
- 通过组合多个源插入来改进来自分布式表的异步插入的性能。要启用此功能，请使用设置 distributed_directory_monitor_batch_inserts=1。

向后不兼容的更改：

- 改变聚合状态的二进制格式 groupArray(array_column) 数组的函数。

更改的完整列表：

- 添加了 output_format_json_quote_denormals 设置，允许以JSON格式输出nan和inf值。
- 从分布式表读取时优化流分配。
- 如果值没有更改，可以在只读模式下配置设置。
- 添加了检索MergeTree引擎的非整数颗粒的功能，以满足preferred_block_size_bytes设置中指定的块大小的限制。其目的是在处理来自具有大列的表的查询时减少RAM消耗并增加缓存局部性。
- 高效使用包含如下表达式的索引 toStartOfHour(x) 对于像条件 toStartOfHour(x) op constexpr。
- 添加了MergeTree引擎的新设置（配置中的merge_tree部分。xml）：
 - replicated_deduplication_window_seconds设置复制表中重复数据删除插入所允许的秒数。
 - cleanup_delay_period设置启动清理以删除过时数据的频率。
 - replicated_can_become_leader可以防止副本成为领导者（并分配合并）。
- 加速清理，从ZooKeeper中删除过时的数据。
- 针对群集DDL查询的多个改进和修复。特别令人感兴趣的是新设置distributed_ddl_task_timeout，它限制了等待群集中服务器响应的时间。如果未在所有主机上执行ddl请求，则响应将包含超时错误，并且请求将以异步模式执行。
- 改进了服务器日志中堆栈跟踪的显示。
- 添加了“none”压缩方法的值。
- 您可以在config中使用多个dictionaries_config部分。xml
- 可以通过文件系统中的套接字连接到MySQL。
- 系统。部件表有一个新的列，其中包含有关标记大小的信息，以字节为单位。

错误修复：

- 使用合并表的分布式表现在可以正确地用于具有条件的SELECT查询 `_table` 场。
- 修复了检查数据部分时ReplicatedMergeTree中罕见的争用条件。
- 固定可能冻结“leader election”启动服务器时。
- 使用数据源的本地副本时，将忽略`max_replica_delay_for_distributed_queries`设置。这已被修复。
- 修正了不正确的行为 `ALTER TABLE CLEAR COLUMN IN PARTITION` 尝试清除不存在的列时。
- 修复了`multif`函数中使用空数组或字符串时的异常。
- 修正了反序列化本机格式时过多的内存分配。
- 修正了Trie字典的不正确的自动更新。
- 修复了使用SAMPLE从合并表中使用GROUP BY子句运行查询时的异常。
- 修复了使用`distributed_aggregation_memory_efficient=1`时组的崩溃。
- 现在，您可以指定数据库。表在右侧的IN和JOIN。
- 用于并行聚合的线程太多。这已被修复。
- 固定如何“if”函数与`FixedString`参数一起使用。
- 为权重为0的分片从分布式表中选择工作不正确。这已被修复。
- 运行 `CREATE VIEW IF EXISTS` no longer causes crashes.
- 修正了`input_format_skip_unknown_fields=1`设置并且有负数时的不正确行为。
- 修正了一个无限循环 `dictGetHierarchy()` 如果字典中有一些无效的数据，则函数。
- 固定 `Syntax error: unexpected (...)` 在IN或JOIN子句和合并表中使用子查询运行分布式查询时出错。
- 修复了从字典表中选择查询的不正确解释。
- 修正了“Cannot mremap”在IN和JOIN子句中使用包含超过20亿个元素的数组时出错。
- 修复了以MySQL为源的字典的故障转移。

改进开发和组装ClickHouse的工作流程:

- 构建可以在阿卡迪亚组装。
- 您可以使用gcc7来编译ClickHouse。
- 现在使用ccache+distcc的并行构建速度更快。

碌莽禄,拢,010-68520682\<url>戮卤箋拢,010-68520682\<url>

新功能:

- 分布式的DDL（例如，`CREATE TABLE ON CLUSTER`）
- 复制的查询 `ALTER TABLE CLEAR COLUMN IN PARTITION.`
- 字典表的引擎（以表格形式访问字典数据）。
- 字典数据库引擎（这种类型的数据库会自动为所有连接的外部字典提供字典表）。
- 您可以通过向源发送请求来检查字典的更新。
- 限定列名称
- 使用双引号引用标识符。
- Http接口中的会话。
- 复制表的优化查询不仅可以在leader上运行。

向后不兼容的更改:

- 删除设置全局。

小的变化:

- 现在，在触发警报之后，日志将打印完整的堆栈跟踪。
- 在启动时放宽对损坏/额外数据部件数量的验证（有太多误报）。

错误修复:

- 修复了连接错误“sticking”当插入到分布式表中。
- GLOBAL IN现在适用于查看分布式表的合并表中的查询。
- 在Google Compute Engine虚拟机上检测到不正确的内核数。这已被修复。
- 缓存外部字典的可执行源如何工作的更改。
- 修复了包含空字符的字符串的比较。
- 修正了Float32主键字段与常量的比较。
- 以前，对字段大小的不正确估计可能导致分配过大。

- 修复了使用ALTER查询添加到表中的可空列时的崩溃。
- 修复了按可空列排序时的崩溃，如果行数小于限制。
- 修复了仅由常量值组成的子查询的顺序。
- 以前，复制的表在丢弃表失败后可能仍处于无效状态。
- 具有空结果的标量子查询的别名不再丢失。
- 现在如果.so文件被损坏，使用编译的查询不会失败并出现错误。

修复于 ClickHouse Release 18.12.13, 2018-09-10

CVE-2018-14672

加载CatBoost模型的功能，允许遍历路径并通过错误消息读取任意文件。

来源: Yandex信息安全部队的Andrey Krasichkov

修复于 ClickHouse Release 18.10.3, 2018-08-13

CVE-2018-14671

unixODBC允许从文件系统加载任意共享对象，从而导致«远程执行代码»漏洞。

来源：Yandex信息安全部队的Andrey Krasichkov和Evgeny Sidorov

修复于 ClickHouse Release 1.1.54388, 2018-06-28

CVE-2018-14668

远程表函数功能允许在 «user», «password» 及 «default_database» 字段中使用任意符号，从而导致跨协议请求伪造攻击。

来源：Yandex信息安全部队的Andrey Krasichkov

修复于 ClickHouse Release 1.1.54390, 2018-07-06

CVE-2018-14669

ClickHouse MySQL客户端启用了 «LOAD DATA LOCAL INFILE» 功能，该功能允许恶意MySQL数据库从连接的ClickHouse服务器读取任意文件。

来源：Yandex信息安全部队的Andrey Krasichkov和Evgeny Sidorov

修复于 ClickHouse Release 1.1.54131, 2017-01-10

CVE-2018-14670

deb软件包中的错误配置可能导致使用未经授权的数据库。

来源：英国国家网络安全中心（NCSC）

规划

Q1 2020

- 更精确的用户资源池，可以在用户之间合理分配集群资源
- 细粒度的授权管理
- 与外部认证服务集成

常见问题

为什么不使用MapReduce之类的产品呢？

我们可以将MapReduce这类的系统称为分布式计算系统，其reduce操作基于分布式排序。其中最常见的开源解决方案是 [Apache Hadoop](#)。Yandex使用他们的内部解决方案YT。

这些系统不适合在线查询，因为它们的延迟高。换句话说，它们不能用作Web接口的后端服务。这些系统对于实时数据更新是没有用的。如果操作的结果和所有中间结果（如果有的话）位于单个服务器的内存中，则分布式排序不是执行reduce操作的最佳方式，但这通常是在线查询的情况。在这种情况下，哈希表是执行reduce操作的最佳方式。优化map-reduce任务的常用方法是使用内存中的哈希表进行预聚合（部分reduce），用户手动执行此优化操作。分布式排序是运行简单map-reduce任务时性能降低的主要原因之一。

大多数MapReduce系统允许您在集群上执行任意代码。但是，声明性查询语言更适合OLAP，以便快速运行实验。例如，Hadoop包含Hive和Pig，Cloudera Impala或Shark（过时）for Spark，以及Spark SQL、Presto和Apache Drill。与专业系统相比，运行此类任务时的性能非常不理想，所以将这些系统用作Web接口的后端服务是不现实的，因为延迟相对较高。

如果我在通过ODBC使用Oracle时遇到编码问题，该怎么办？

如果您通过ODBC驱动程序使用Oracle作为外部字典的源，则需要为 `NLS_LANG` 在变量 `/etc/default/clickhouse`。欲了解更多详情，请参阅 [Oracle NLS_常见问题](#).

示例

```
NLS_LANG=CHINESE_CHINA.ZHS16GBK
```

浏览ClickHouse源代码

您可以使用 [Woboq](#) 在线代码浏览器可用 [这里](#). 它提供了代码导航和语义突出显示，搜索和索引。代码快照每天更新。

此外，您还可以浏览源 [GitHub](#) 像往常一样

如果你有兴趣使用什么样的IDE，我们建议CLion，QT Creator，VS Code和KDevelop（有注意事项）。您可以使用任何喜欢的IDE。Vim和Emacs也算数。

如何在Linux上为AARCH64（ARM64）架构构建ClickHouse

这是当你有Linux机器，并希望使用它来构建的情况下 `clickhouse` 二进制文件将运行在另一个Linux机器上与AARCH64CPU架构。这适用于在Linux服务器上运行的持续集成检查。

Aarch64的交叉构建基于 [构建说明](#) 先跟着他们

安装Clang-8

按照以下说明操作<https://apt.llvm.org/>为您的Ubuntu或Debian设置.

例如，在Ubuntu Bionic中，您可以使用以下命令:

```
echo "deb [trusted=yes] http://apt.llvm.org/bionic/ llvm-toolchain-bionic-8 main" | sudo tee  
/etc/apt/sources.list.d/llvm.list  
sudo apt-get update  
sudo apt-get install clang-8
```

安装交叉编译工具集

```
cd ClickHouse
```

```
mkdir -p build-aarch64/cmake/toolchain/linux-aarch64
wget 'https://developer.arm.com/-/media/Files/downloads/gnu-a/8.3-2019.03/binrel/gcc-arm-8.3-2019.03-x86_64-
aarch64-linux-gnu.tar.xz?revision=2e88a73f-d233-4f96-b1f4-d8b36e9bb0b9&la=en' -O gcc-arm-8.3-2019.03-x86_64-
aarch64-linux-gnu.tar.xz
tar xJf gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu.tar.xz -C build-aarch64/cmake/toolchain/linux-aarch64 --strip-
components=1
```

建立ClickHouse

```
cd ClickHouse
mkdir build-arm64
CC=clang-8 CXX=clang++-8 cmake . -Bbuild-arm64 -DCMAKE_TOOLCHAIN_FILE=cmake/linux/toolchain-aarch64.cmake
ninja -C build-arm64
```

生成的二进制文件将仅在具有AARCH64CPU体系结构的Linux上运行。

ClickHouse 开发

ClickHouse 架构概述

ClickHouse 是一个真正的列式数据库管理系统（DBMS）。在 ClickHouse 中，数据始终是按列存储的，包括矢量（向量或列块）执行的过程。只要有可能，操作都是基于矢量进行分派的，而不是单个的值，这被称为《矢量化查询执行》，它有利于降低实际的数据处理开销。

这个想法并不新鲜，其可以追溯到 APL 编程语言及其后代：A+、J、K 和 Q。矢量编程被大量用于科学数据处理中。即使在关系型数据库中，这个想法也不是什么新的东西：比如，矢量编程也被大量用于 Vectorwise 系统中。

通常有两种不同的加速查询处理的方法：矢量化查询执行和运行时代码生成。在后者中，动态地为每一类查询生成代码，消除了间接分派和动态分派。这两种方法中，并没有哪一种严格地比另一种好。运行时代码生成可以更好地将多个操作融合在一起，从而充分利用 CPU 执行单元和流水线。矢量化查询执行不是特别实用，因为它涉及必须写到缓存并读回的临时向量。如果 L2 缓存容纳不下临时数据，那么这将成为一个问题。但矢量化查询执行更容易利用 CPU 的 SIMD 功能。朋友写的一篇[研究论文](#)表明，将两种方法结合起来是更好的选择。ClickHouse 使用了矢量化查询执行，同时初步提供了有限的运行时动态代码生成。

列（Columns）

要表示内存中的列（实际上是列块），需使用 `IColumn` 接口。该接口提供了用于实现各种关系操作符的辅助方法。几乎所有的操作都是不可变的：这些操作不会更改原始列，但是会创建一个新的修改后的列。比如，`IColumn::filter` 方法接受过滤字节掩码，用于 `WHERE` 和 `HAVING` 关系操作符中。另外的例子：`IColumn::permute` 方法支持 `ORDER BY` 实现，`IColumn::cut` 方法支持 `LIMIT` 实现等等。

不同的 `IColumn` 实现（`ColumnUInt8`、`ColumnString` 等）负责不同的列内存布局。内存布局通常是一个连续的数组。对于数据类型为整型的列，只是一个连续的数组，比如 `std::vector`。对于 `String` 列和 `Array` 列，则由两个向量组成：其中一个向量连续存储所有的 `String` 或数组元素，另一个存储每一个 `String` 或 `Array` 的起始元素在第一个向量中的偏移。而 `ColumnConst` 则仅在内存中存储一个值，但是看起来像一个列。

字段

尽管如此，有时候也可能需要处理单个值。表示单个值，可以使用 `Field`。`Field` 是 `UInt64`、`Int64`、`Float64`、`String` 和 `Array` 组成的联合。`IColumn` 拥有 `operator[]` 方法来获取第 `n` 个值成为一个 `Field`，同时也拥有 `insert` 方法将一个 `Field` 追加到一个列的末尾。这些方法并不高效，因为它们需要处理表示单一值的临时 `Field` 对象，但是有更高效的方法比如 `insertFrom` 和 `insertRangeFrom` 等。

`Field` 中并没有足够的关于一个表（table）的特定数据类型的信息。比如，`UInt8`、`UInt16`、`UInt32` 和 `UInt64` 在 `Field` 中均表示为 `UInt64`。

抽象漏洞

`IColumn` 具有用于数据的常见关系转换的方法，但这些方法并不能够满足所有需求。比如，`ColumnUInt64` 没有用于计算两列和的方法，`ColumnString` 没有用于进行子串搜索的方法。这些无法计算的例程在 `Icolumn` 之外实现。

列 (`Columns`) 上的各种函数可以通过使用 `Icolumn` 的方法来提取 `Field` 值，或根据特定的 `Icolumn` 实现的数据内存布局的知识，以一种通用但不高效的方式实现。为此，函数将会转换为特定的 `IColumn` 类型并直接处理内部表示。比如，`ColumnUInt64` 具有 `getData` 方法，该方法返回一个指向列的内部数组的引用，然后一个单独的例程可以直接读写或填充该数组。实际上，《抽象漏洞 (leaky abstractions)》允许我们以更高效的方式来实现各种特定的例程。

数据类型

`IDataType` 负责序列化和反序列化：读写二进制或文本形式的列或单个值构成的块。`IDataType` 直接与表的数据类型相对应。比如，有 `DataTypeUInt32`、`DataTypeDateTime`、`DataTypeString` 等数据类型。

`IDataType` 与 `IColumn` 之间的关联并不大。不同的数据类型在内存中能够用相同的 `IColumn` 实现来表示。比如，`DataTypeUInt32` 和 `DataTypeDateTime` 都是用 `ColumnUInt32` 或 `ColumnConstUInt32` 来表示的。另外，相同的数据类型也可以用不同的 `IColumn` 实现来表示。比如，`DataTypeUInt8` 既可以使用 `ColumnUInt8` 来表示，也可以使用过 `ColumnConstUInt8` 来表示。

`IDataType` 仅存储元数据。比如，`DataTypeUInt8` 不存储任何东西（除了 `vptr`）；`DataTypeFixedString` 仅存储 `N`（固定长度字符串的串长度）。

`IDataType` 具有针对性各种数据格式的辅助函数。比如如下一些辅助函数：序列化一个值并加上可能的引号；序列化一个值用于 JSON 格式；序列化一个值作为 XML 格式的一部分。辅助函数与数据格式并没有直接的对应。比如，两种不同的数据格式 `Pretty` 和 `TabSeparated` 均可以使用 `IDataType` 接口提供的 `serializeTextEscaped` 这一辅助函数。

块 (Block)

`Block` 是表示内存中表的子集 (chunk) 的容器，是由三元组：`(IColumn, IDatatype, 列名)` 构成的集合。在查询执行期间，数据是按 `Block` 进行处理的。如果我们有一个 `Block`，那么就有了数据（在 `IColumn` 对象中），有了数据的类型信息告诉我们如何处理该列，同时也有了列名（来自表的原始列名，或人为指定的用于临时计算结果的名字）。

当我们遍历一个块中的列进行某些函数计算时，会把结果列加入到块中，但不会更改函数参数中的列，因为操作是不可变的。之后，不需要的列可以从块中删除，但不是修改。这对于消除公共子表达式非常方便。

`Block` 用于处理数据块。注意，对于相同类型的计算，列名和类型对不同的块保持相同，仅列数据不同。最好把块数据 (`block data`) 和块头 (`block header`) 分离开来，因为小块大小会因复制共享指针和列名而带来很高的临时字符串开销。

块流 (Block Streams)

块流用于处理数据。我们可以使用块流从某个地方读取数据，执行数据转换，或将数据写到某个地方。`IBlockInputStream` 具有 `read` 方法，其能够在数据可用时获取下一个块。`IBlockOutputStream` 具有 `write` 方法，其能够将块写到某处。

块流负责：

1. 读或写一个表。表仅返回一个流用于读写块。
2. 完成数据格式化。比如，如果你打算将数据以 `Pretty` 格式输出到终端，你可以创建一个块输出流，将块写入该流中，然后进行格式化。
3. 执行数据转换。假设你现在有 `IBlockInputStream` 并且打算创建一个过滤流，那么你可以创建一个 `FilterBlockInputStream` 并用 `IBlockInputStream` 进行初始化。之后，当你从 `FilterBlockInputStream` 中拉取块时，会从你的流中提取一个块，对其进行过滤，然后将过滤后的块返回给你。查询执行流水线就是以这种方式表示的。

还有一些更复杂的转换。比如，当你从 `AggregatingBlockInputStream` 拉取数据时，会从数据源读取全部数据进行聚集，然后将聚集后的数据流返回给你。另一个例子：`UnionBlockInputStream` 的构造函数接受多个输入源和多个线程，其能够启动多线程从多个输入源并行读取数据。

块流使用《pull》方法来控制流：当你从第一个流中拉取块时，它会接着从嵌套的流中拉取所需的块，然后整个执行流水线开始工作。《pull》和《push》都不是最好的方案，因为控制流不是明确的，这限制了各种功能的实现，比如多个查询同步执行（多个流水线合并到一起）。这个限制可以通过协程或直接运行互相等待的线程来解决。如果控制流明确，那么我们会有更多的可能性：如果我们定位了数据从一个计算单元传递到那些外部的计算单元中其中一个计算单元的逻辑。阅读这篇[文章](#)来

获取更多的想法。

我们需要注意，查询执行流水线在每一步都会创建临时数据。我们要尽量使块的大小足够小，从而 CPU 缓存能够容纳下临时数据。在这个假设下，与其他计算相比，读写临时数据几乎是没有任何开销的。我们也可以考虑一种替代方案：将流水线中的多个操作融合在一起，使流水线尽可能短，并删除大量临时数据。这可能是一个优点，但同时也有缺点。比如，拆分流水线使得中间数据缓存、获取同时运行的类似查询的中间数据以及相似查询的流水线合并等功能很容易实现。

格式 (Formats)

数据格式同块流一起实现。既有仅用于向客户端输出数据的»展示«格式，如 `IBlockOutputStream` 提供的 `Pretty` 格式，也有其它输入输出格式，比如 `TabSeparated` 或 `JSONEachRow`。

此外还有行流：`IRowInputStream` 和 `IRowOutputStream`。它们允许你按行 `pull/push` 数据，而不是按块。行流只需要简单地面向行格式实现。包装器 `BlockInputStreamFromRowInputStream` 和 `BlockOutputStreamFromRowOutputStream` 允许你将面向行的流转换为正常的面向块的流。

I/O

对于面向字节的输入输出，有 `ReadBuffer` 和 `WriteBuffer` 这两个抽象类。它们用来替代 C++ 的 `iostream`。不用担心：每个成熟的 C++ 项目都会有充分的理由使用某些东西来代替 `iostream`。

`ReadBuffer` 和 `WriteBuffer` 由一个连续的缓冲区和指向缓冲区中某个位置的一个指针组成。实现中，缓冲区可能拥有内存，也可能不拥有内存。有一个虚方法会使用随后的数据来填充缓冲区（针对 `ReadBuffer`）或刷新缓冲区（针对 `WriteBuffer`），该虚方法很少被调用。

`ReadBuffer` 和 `WriteBuffer` 的实现用于处理文件、文件描述符和网络套接字（socket），也用于实现压缩（`CompressedWriteBuffer` 在写入数据前需要先用一个 `WriteBuffer` 进行初始化并进行压缩）和其它用途。`ConcatReadBuffer`、`LimitReadBuffer` 和 `HashingWriteBuffer` 的用途正如其名字所描述的一样。

`ReadBuffer` 和 `WriteBuffer` 仅处理字节。为了实现格式化输入和输出（比如以十进制格式写一个数字），`ReadHelpers` 和 `WriteHelpers` 头文件中有一些辅助函数可用。

让我们来看一下，当你把一个结果集以 JSON 格式写到标准输出 (`stdout`) 时会发生什么。你已经准备好从 `IBlockInputStream` 获取结果集，然后创建 `WriteBufferFromFileDescriptor(STDOUT_FILENO)` 用于写字节到标准输出，创建 `JSONRowOutputStream` 并用 `WriteBuffer` 初始化，用于将行以 JSON 格式写到标准输出，你还可以在其上创建 `BlockOutputStreamFromRowOutputStream`，将其表示为 `IBlockOutputStream`。然后调用 `copyData` 将数据从 `IBlockInputStream` 传输到 `IBlockOutputStream`，一切工作正常。在内部，`JSONRowOutputStream` 会写入 JSON 分隔符，并以指向 `IColumn` 的引用和行数作为参数调用 `IDataType::serializeTextJSON` 函数。随后，`IDataType::serializeTextJSON` 将会调用 `WriteHelpers.h` 中的一个方法：比如，`writeText` 用于数值类型，`writeJSONString` 用于 `DataTypeString`。

表 (Tables)

表由 `IStorage` 接口表示。该接口的不同实现对应不同的表引擎。比如 `StorageMergeTree`、`StorageMemory` 等。这些类的实例就是表。

`IStorage` 中最重要的方法是 `read` 和 `write`，除此之外还有 `alter`、`rename` 和 `drop` 等方法。`read` 方法接受如下参数：需要从表中读取的列集，需要执行的 AST 查询，以及所需返回的流的数量。`read` 方法的返回值是一个或多个 `IBlockInputStream` 对象，以及在查询执行期间在一个表引擎内完成的关于数据处理阶段的信息。

在大多数情况下，`read` 方法仅负责从表中读取指定的列，而不会进行进一步的数据处理。进一步的数据处理均由查询解释器完成，不由 `IStorage` 负责。

但是也有值得注意的例外：

- AST 查询被传递给 `read` 方法，表引擎可以使用它来判断是否能够使用索引，从而从表中读取更少的数据。
- 有时候，表引擎能够将数据处理到一个特定阶段。比如，`StorageDistributed` 可以向远程服务器发送查询，要求它们将来自不同的远程服务器能够合并的数据处理到某个阶段，并返回预处理后的数据，然后查询解释器完成后续的数据处理。

表的 `read` 方法能够返回多个 `IBlockInputStream` 对象以允许并行处理数据。多个块输入流能够从一个表中并行读取。然后你可以通过不同的转换对这些流进行装饰（比如表达式求值或过滤），转换过程能够独立计算，并在其上创建一个 `UnionBlockInputStream`，以并行读取多个流。

另外也有 `TableFunction`。`TableFunction` 能够在查询的 `FROM` 字句中返回一个临时的 `IStorage` 以供使用。

要快速了解如何实现自己的表引擎，可以查看一些简单的表引擎，比如 `StorageMemory` 或 `StorageTinyLog`。

作为 `read` 方法的结果，`IStorage` 返回 `QueryProcessingStage` - 关于 `storage` 里哪部分查询已经被计算的信息。当前我们仅有非常粗粒度的信息。`Storage` 无法告诉我们«对于这个范围的数据，我已经处理完了 `WHERE` 字句里的这部分表达式»。我们需要在这个地方继续努力。

解析器 (Parsers)

查询由一个手写递归下降解析器解析。比如，`ParserSelectQuery` 只是针对查询的不同部分递归地调用下层解析器。解析器创建 `AST`。`AST` 由节点表示，节点是 `IAST` 的实例。

由于历史原因，未使用解析器生成器。

解释器 (Interpreters)

解释器负责从 `AST` 创建查询执行流水线。既有一些简单的解释器，如 `InterpreterExistsQuery` 和 `InterpreterDropQuery`，也有更复杂的解释器，如 `InterpreterSelectQuery`。查询执行流水线由块输入或输出流组成。比如，`SELECT` 查询的解释结果是从 `FROM` 字句的结果集中读取数据的 `IBlockInputStream`；`INSERT` 查询的结果是写入需要插入的数据的 `IBlockOutputStream`；`SELECT INSERT` 查询的解释结果是 `IBlockInputStream`，它在第一次读取时返回一个空结果集，同时将数据从 `SELECT` 复制到 `INSERT`。

`InterpreterSelectQuery` 使用 `ExpressionAnalyzer` 和 `ExpressionActions` 机制来进行查询分析和转换。这是大多数基于规则的查询优化完成的地方。`ExpressionAnalyzer` 非常混乱，应该进行重写：不同的查询转换和优化应该被提取出来并划分成不同的类，从而允许模块化转换或查询。

函数 (Functions)

函数既有普通函数，也有聚合函数。对于聚合函数，请看下一节。

普通函数不会改变行数 - 它们的执行看起来就像是独立地处理每一行数据。实际上，函数不会作用于一个单独的行上，而是作用在以 `Block` 为单位的数据上，以实现向量查询执行。

还有一些杂项函数，比如 **块大小**、**rowNumberInBlock**，以及 **跑累积**，它们对块进行处理，并且不遵从行的独立性。

ClickHouse 具有强类型，因此隐式类型转换不会发生。如果函数不支持某个特定的类型组合，则会抛出异常。但函数可以通过重载以支持许多不同的类型组合。比如，`plus` 函数（用于实现 `+` 运算符）支持任意数字类型的组合：`UInt8 + Float32`，`UInt16 + Int8` 等。同时，一些可变参数的函数能够级接收任意数目的参数，比如 `concat` 函数。

实现函数可能有些不方便，因为函数的实现需要包含所有支持该操作的数据类型和 `IColumn` 类型。比如，`plus` 函数能够利用 C++ 模板针对不同的数字类型组合、常量以及非常量的左值和右值进行代码生成。

这是一个实现动态代码生成的好地方，从而能够避免模板代码膨胀。同样，运行时代码生成也使得实现融合函数成为可能，比如融合《乘-加》，或者在单层循环迭代中进行多重比较。

由于向量查询执行，函数不会《短路》。比如，如果你写 `WHERE f(x) AND g(y)`，两边都会进行计算，即使是对于 `f(x)` 为 0 的行（除非 `f(x)` 是零常量表达式）。但是如果 `f(x)` 的选择条件很高，并且计算 `f(x)` 比计算 `g(y)` 要划算得多，那么最好进行多遍计算：首先计算 `f(x)`，根据计算结果对列数据进行过滤，然后计算 `g(y)`，之后只需对较小数量的数据进行过滤。

聚合函数

聚合函数是状态函数。它们将传入的值激活到某个状态，并允许你从该状态获取结果。聚合函数使用 `IAggregateFunction` 接口进行管理。状态可以非常简单 (`AggregateFunctionCount` 的状态只是一个单一的 `UInt64` 值)，也可以非常复杂 (`AggregateFunctionUnionCombined` 的状态是由一个线性数组、一个散列表和一个 `HyperLogLog` 概率数据结构组合而成)。

的）。

为了能够在执行一个基数很大的 `GROUP BY` 查询时处理多个聚合状态，需要在 `Arena`（一个内存池）或任何合适的内存块中分配状态。状态可以有一个非平凡的构造器和析构器：比如，复杂的聚合状态能够自己分配额外的内存。这需要注意状态的创建和销毁并恰当地传递状态的所有权，以跟踪谁将何时销毁状态。

聚合状态可以被序列化和反序列化，以在分布式查询执行期间通过网络传递或者在内存不够的时候将其写到硬盘。聚合状态甚至可以通过 `DataTypeAggregateFunction` 存储到一个表中，以允许数据的增量聚合。

聚合函数状态的序列化数据格式目前尚未版本化。如果只是临时存储聚合状态，这样是可以的。但是我们有 `AggregatingMergeTree` 表引擎用于增量聚合，并且人们已经在生产中使用它。这就是为什么在未来当我们更改任何聚合函数的序列化格式时需要增加向后兼容的支持。

服务器 (Server)

服务器实现了多个不同的接口：

- 一个用于任何外部客户端的 `HTTP` 接口。
- 一个用于本机 `ClickHouse` 客户端以及在分布式查询执行中跨服务器通信的 `TCP` 接口。
- 一个用于传输数据以进行拷贝的接口。

在内部，它只是一个没有协程、纤程等的基础多线程服务器。服务器不是为处理高速率的简单查询设计的，而是为处理相对低速率的复杂查询设计的，每一个复杂查询能够对大量的数据进行处理分析。

服务器使用必要的查询执行需要的环境初始化 `Context` 类：可用数据库列表、用户和访问权限、设置、集群、进程列表和查询日志等。这些环境被解释器使用。

我们维护了服务器 `TCP` 协议的完全向后向前兼容性：旧客户端可以和新服务器通信，新客户端也可以和旧服务器通信。但是我们并不想永久维护它，我们将在大约一年后删除对旧版本的支持。

对于所有的外部应用，我们推荐使用 `HTTP` 接口，因为该接口很简单，容易使用。`TCP` 接口与内部数据结构的联系更加紧密：它使用内部格式传递数据块，并使用自定义帧来压缩数据。我们没有发布该协议的 C 库，因为它需要链接大部分的 `ClickHouse` 代码库，这是不切实际的。

分布式查询执行

集群设置中的服务器大多是独立的。你可以在一个集群中的一个或多个服务器上创建一个 `Distributed` 表。`Distributed` 表本身并不存储数据，它只为集群的多个节点上的所有本地表提供一个«视图 (view)»。当从 `Distributed` 表中进行 `SELECT` 时，它会重写该查询，根据负载平衡设置来选择远程节点，并将查询发送给节点。`Distributed` 表请求远程服务器处理查询，直到可以合并来自不同服务器的中间结果的阶段。然后它接收中间结果并进行合并。分布式表会尝试将尽可能多的工作分配给远程服务器，并且不会通过网络发送太多的中间数据。

当 `IN` 或 `JOIN` 子句中包含子查询并且每个子查询都使用分布式表时，事情会变得更加复杂。我们有不同的策略来执行这些查询。

分布式查询执行没有全局查询计划。每个节点都有针对自己的工作部分的本地查询计划。我们仅有简单的一次性分布式查询执行：将查询发送给远程节点，然后合并结果。但是对于具有高基数的 `GROUP BY` 或具有大量临时数据的 `JOIN` 这样困难的查询来说，这是不可行的：在这种情况下，我们需要在服务器之间«改组»数据，这需要额外的协调。`ClickHouse` 不支持这类查询执行，我们需要在这方面进行努力。

合并树

`MergeTree` 是一系列支持按主键索引的存储引擎。主键可以是一个任意的列或表达式的元组。`MergeTree` 表中的数据存储于«分块»中。每一个分块以主键序存储数据（数据按主键元组的字典序排序）。表的所有列都存储在这些«分块»中分离的 `column.bin` 文件中。`column.bin` 文件由压缩块组成，每一个块通常是 64 KB 到 1 MB 大小的未压缩数据，具体取决于平均值大小。这些块由一个接一个连续放置的列值组成。每一列的列值顺序相同（顺序由主键定义），因此当你按多列进行迭代时，你能够得到相应列的值。

主键本身是《稀疏》的。它并不是索引单一的行，而是索引某个范围内的数据。一个单独的 `primary.idx` 文件具有每个第 N 行的主键值，其中 N 称为 `index_granularity`（通常，N = 8192）。同时，对于每一列，都有带有标记的 `column.mrk` 文件，该文件记录的是每个第 N 行在数据文件中的偏移量。每个标记是一个 pair：文件中的偏移量到压缩块的起始，以及解压缩块中的偏移量到数据的起始。通常，压缩块根据标记对齐，并且解压缩块中的偏移量为 0。`primary.idx` 的数据始终驻留在内存，同时 `column.mrk` 的数据被缓存。

当我们要从 `MergeTree` 的一个分块中读取部分内容时，我们会查看 `primary.idx` 数据并查找可能包含所请求数据的范围，然后查看 `column.mrk` 并计算偏移量从而得知从哪里开始读取些范围的数据。由于稀疏性，可能会读取额外的数据。

`ClickHouse` 不适用于高负载的简单点查询，因为对于每一个键，整个 `index_granularity` 范围的行的数据都需要读取，并且对于每一列需要解压缩整个压缩块。我们使索引稀疏，是因为每一个单一的服务器需要在索引没有明显内存消耗的情况下，维护数万亿行的数据。另外，由于主键是稀疏的，导致其不是唯一的：无法在 `INSERT` 时检查一个键在表中是否存在。你可以在一个表中使用同一个键创建多个行。

当你向 `MergeTree` 中插入一堆数据时，数据按主键排序并形成一个新的分块。为了保证分块的数量相对较少，有后台线程定期选择一些分块并将它们合并成一个有序的分块，这就是 `MergeTree` 的名称来源。当然，合并会导致《写入放大》。所有的分块都是不可变的：它们仅会被创建和删除，不会被修改。当运行 `SELECT` 查询时，`MergeTree` 会保存一个表的快照（分块集合）。合并之后，还会保留旧的分块一段时间，以便发生故障后更容易恢复，因此如果我们发现某些合并后的分块可能已损坏，我们可以将其替换为原分块。

`MergeTree` 不是 `LSM` 树，因为它不包含《memtable》和《log》：插入的数据直接写入文件系统。这使得它仅适用于批量插入数据，而不适用于非常频繁地一行一行插入 - 大约每秒一次是没问题的，但是每秒一千次就会有问题。我们这样做是为了简单起见，因为我们已经在我们的应用中批量插入数据。

`MergeTree` 表只能有一个（主）索引：没有任何辅助索引。在一个逻辑表下，允许有多个物理表示，比如，可以以多个物理顺序存储数据，或者同时表示预聚合数据和原始数据。

有些 `MergeTree` 引擎会在后台合并期间做一些额外工作，比如 `CollapsingMergeTree` 和 `AggregatingMergeTree`。这可以视为对更新的特殊支持。请记住这些不是真正的更新，因为用户通常无法控制后台合并将会执行的时间，并且 `MergeTree` 中的数据几乎总是存储在多个分块中，而不是完全合并的形式。

复制（Replication）

`ClickHouse` 中的复制是基于表实现的。你可以在同一个服务器上有一些可复制的表和不可复制的表。你也可以以不同的方式进行表的复制，比如一个表进行双因子复制，另一个进行三因子复制。

复制是在 `ReplicatedMergeTree` 存储引擎中实现的。`ZooKeeper` 中的路径被指定为存储引擎的参数。`ZooKeeper` 中所有具有相同路径的表互为副本：它们同步数据并保持一致性。只需创建或删除表，就可以实现动态添加或删除副本。

复制使用异步多主机方案。你可以将数据插入到与 `ZooKeeper` 进行会话的任意副本中，并将数据复制到所有其它副本中。由于 `ClickHouse` 不支持 `UPDATEs`，因此复制是无冲突的。由于没有对插入的仲裁确认，如果一个节点发生故障，刚刚插入的数据可能会丢失。

用于复制的元数据存储在 `ZooKeeper` 中。其中一个复制日志列出了要执行的操作。操作包括：获取分块、合并分块和删除分区等。每一个副本将复制日志复制到其队列中，然后执行队列中的操作。比如，在插入时，在复制日志中创建《获取分块》这一操作，然后每一个副本都会去下载该分块。所有副本之间会协调进行合并以获得相同字节的结果。所有的分块在所有的副本上以相同的方式合并。为实现该目的，其中一个副本被选为领导者，该副本首先进行合并，并把《合并分块》操作写到日志中。

复制是物理的：只有压缩的分块会在节点之间传输，查询则不会。为了降低网络成本（避免网络放大），大多数情况下，会在每一个副本上独立地处理合并。只有在存在显著的合并延迟的情况下，才会通过网络发送大块的合并分块。

另外，每一个副本将其状态作为分块和校验和组成的集合存储在 `ZooKeeper` 中。当本地文件系统中的状态与 `ZooKeeper` 中引用的状态不同时，该副本会通过从其它副本下载缺失和损坏的分块来恢复其一致性。当本地文件系统中出现一些意外或损坏的数据时，`ClickHouse` 不会将其删除，而是将其移动到一个单独的目录下并忘记它。

`ClickHouse` 集群由独立的分片组成，每一个分片由多个副本组成。集群不是弹性的，因此在添加新的分片后，数据不会自动在分片之间重新平衡。相反，集群负载将变得不均衡。该实现为你提供了更多控制，对于相对较小的集群，例如只有数十个节点的集群也沿用旧机制。一旦对于我们的大部分由伟图的项目来说无人关注的集群来说，这种设计就不再适用。我们应

所有的表都是从单个表中派生出来的。但是，我们正在生产下使用的共有数目的所有的表，这样才形成一个更大的数据库。我们应该实现一个表引擎，使得该引擎能够跨集群扩展数据，同时具有动态复制的区域，这些区域能够在集群之间自动拆分和平衡。

ClickHouse 测试

功能性测试

功能性测试是最简便使用的。绝大部分 ClickHouse 的功能可以通过功能性测试来测试，任何代码的更改都必须通过该测试。

每个功能测试会向正在运行的 ClickHouse 服务器发送一个或多个查询，并将结果与预期结果进行比较。

测试用例在 `tests/queries` 目录中。这里有两个子目录：`stateless` 和 `stateful` 目录。无状态的测试无需预加载测试数据集 - 通常是在测试运行期间动态创建小量的数据集。有状态测试需要来自 Yandex.Metrica 的预加载测试数据，而不向一般公众提供。我们倾向于仅使用《无状态》测试并避免添加新的《有状态》测试。

每个测试用例可以是两种类型之一：`.sql` 和 `.sh`。`.sql` 测试文件是用于管理 `clickhouse-client --multiquery --testmode` 的简单 SQL 脚本。`.sh` 测试文件是一个可以自己运行的脚本。

要运行所有测试，请使用 `tests/clickhouse-test` 工具，用 `--help` 可以获取所有的选项列表。您可以简单地运行所有测试或运行测试名称中的子字符串过滤的测试子集：`./clickhouse-test substring`。

调用功能测试最简单的方法是将 `clickhouse-client` 复制到 `/usr/bin/`，运行 `clickhouse-server`，然后从自己的目录运行 `./clickhouse-test`。

要添加新测试，请在 `tests/queries/0_stateless` 目录内添加新的 `.sql` 或 `.sh` 文件，手动检查，然后按以下方式生成 `.reference` 文件：`clickhouse-client -n --testmode < 00000_test.sql > 00000_test.reference` 或 `./00000_test.sh > ./00000_test.reference`。

测试应该只使用（创建，删除等）`test` 数据库中的表，这些表假定是事先创建的；测试也可以使用临时表。

如果要在功能测试中使用分布式查询，可以利用 `remote` 表函数和 `127.0.0.{1..2}` 地址为服务器查询自身；或者您可以在服务器配置文件中使用预定义的测试集群，例如 `test_shard_localhost`。

有些测试在名称中标有 `zookeeper`，`shard` 或 `long`。`zookeeper` 用于使用 ZooKeeper 的测试；`shard` 用于需要服务器监听 `127.0.0.*` 的测试。`long` 适用于运行时间稍长一秒的测试。

已知的 bug

如果我们知道一些可以通过功能测试轻松复制的错误，我们将准备好的功能测试放在 `tests/queries/bugs` 目录中。当修复错误时，这些测试将被移动到 `tests/queries/0_stateless` 目录中。

集成测试

集成测试允许在集群配置中测试 ClickHouse，并与其他服务器（如 MySQL，Postgres，MongoDB）进行 ClickHouse 交互。它们可用于模拟网络拆分，数据包丢弃等。这些测试在 Docker 下运行，并使用各种软件创建多个容器。

参考 `tests/integration/README.md` 文档关于如何使用集成测试。

请注意，ClickHouse 与第三方驱动程序的集成未经过测试。此外，我们目前还没有与 JDBC 和 ODBC 驱动程序进行集成测试。

单元测试

当您想要测试整个 ClickHouse，而不是单个独立的库或类时，单元测试非常有用。您可以使用 `ENABLE_TESTS` CMake 选项启用或禁用测试构建。单元测试（和其他测试程序）位于代码中的 `tests` 子目录中。要运行单元测试，请键入 `ninja test`。有些测试使用 `gtest`，但有些只是在测试失败时返回非零状态码。

如果代码已经被功能测试覆盖（并且功能测试通常使用起来要简单得多），则不一定要进行单元测试。

性能测试

性能测试允许测量和比较综合查询中 ClickHouse 的某些独立部分的性能。测试位于 `tests/performance` 目录中。每个测试都由 `.xml` 文件表示，并附有测试用例的描述。使用 `clickhouse performance-test` 工具（嵌入在 `clickhouse` 二进制文件中）运行测试。请参阅 `--help` 以进行调用。

每个测试在循环中运行一个或多个查询（可能带有参数组合），并具有一些停止条件（如«最大执行速度不会在三秒内更改»）并测量一些有关查询性能的指标（如«最大执行速度»）。某些测试可以包含预加载的测试数据集的前提条件。

如果要在某些情况下提高 ClickHouse 的性能，并且如果可以在简单查询上观察到改进，则强烈建议编写性能测试。在测试过程中使用 `perf top` 或其他 `perf` 工具总是有意义的。

性能测试不是基于每个提交运行的。不收集性能测试结果，我们手动比较它们。

测试工具和脚本

`tests` 目录中的一些程序不是准备测试，而是测试工具。例如，对于 `Lexer`，有一个工具 `src/Parsers/tests/lexer` 标准输出。您可以使用这些工具作为代码示例以及探索和手动测试。

您还可以将一对文件 `.sh` 和 `.reference` 与工具放在一些预定义的输入上运行它 - 然后可以将脚本结果与 `.reference` 文件进行比较。这些测试不是自动化的。

杂项测试

有一些外部字典的测试位于 `tests/external_dictionaries`，机器学习模型在 `tests/external_models` 目录。这些测试未更新，必须转移到集成测试。

对于分布式数据的插入，有单独的测试。此测试在单独的服务器上运行 ClickHouse 集群并模拟各种故障情况：网络拆分，数据包丢弃（ClickHouse 节点之间，ClickHouse 和 ZooKeeper 之间，ClickHouse 服务器和客户端之间等），进行 `kill -9`，`kill -STOP` 和 `kill -CONT` 等操作，类似 **Jepsen**。然后，测试检查是否已写入所有已确认的插入，并且所有已拒绝的插入都未写入。

在 ClickHouse 开源之前，分布式测试是由单独的团队编写的，但该团队不再使用 ClickHouse，测试是在 Java 中意外编写的。由于这些原因，必须重写分布式测试并将其移至集成测试。

手动测试

当您开发了新的功能，做手动测试也是合理的。可以按照以下步骤来进行：

编译 ClickHouse。在命令行中运行 ClickHouse：进入 `programs clickhouse-server` 目录并运行 `./clickhouse-server`。它会默认使用当前目录的配置文件 (`config.xml`，`users.xml` 以及在 `config.d` 和 `users.d` 目录的文件)。可以使用 `programs clickhouse-client clickhouse-client` 来连接数据库。

或者，您可以安装 ClickHouse 软件包：从 Yandex 存储库中获得稳定版本，或者您可以在 ClickHouse 源根目录中使用 `./release` 构建自己的软件包。然后使用 `sudo service clickhouse-server start` 启动服务器（或停止服务器）。在 `/etc/clickhouse-server/clickhouse-server.log` 中查找日志。

当您的系统上已经安装了 ClickHouse 时，您可以构建一个新的 `clickhouse` 二进制文件并替换现有的二进制文件：

```
sudo service clickhouse-server stop
sudo cp ./clickhouse /usr/bin/
sudo service clickhouse-server start
```

您也可以停止 `clickhouse-server` 并使用相同的配置运行您自己的服务器，日志打印到终端：

```
sudo service clickhouse-server stop
sudo -u clickhouse /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

使用 `gdb` 的一个示例：

```
sudo -u clickhouse gdb --args /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

如果 `clickhouse-server` 已经运行并且您不想停止它，您可以更改 `config.xml` 中的端口号（或在 `config.d` 目录中的文件中覆盖它们），配置适当的数据路径，然后运行它。

`clickhouse` 二进制文件几乎没有依赖关系，适用于各种 Linux 发行版。要快速地测试服务器上的更改，您可以简单地将新建的 `clickhouse` 二进制文件 `scp` 到其他服务器，然后按照上面的示例运行它。

测试环境

在将版本发布为稳定之前，我们将其部署在测试环境中。测试环境是一个处理[Yandex.Metrica] (<https://metrica.yandex.com/>) 总数据的1/39部分大小的集群。我们与 Yandex.Metrica 团队公用我们的测试环境。ClickHouse 在现有数据的基础上无需停机即可升级。我们首先看到数据处理成功而不会实时滞后，复制继续工作，并且 Yandex.Metrica 团队无法看到问题。首先的检查可以通过以下方式完成：

```
SELECT hostName() AS h, any(version()), any(uptime()), max(UTCEventTime), count() FROM remote('example01-01-{1..3}t', merge, hits) WHERE EventDate >= today() - 2 GROUP BY h ORDER BY h;
```

在某些情况下，我们还部署到 Yandex 的合作团队的测试环境：市场，云等。此外，我们还有一些用于开发目的的硬件服务器。

负载测试

部署到测试环境后，我们使用生产群集中的查询运行负载测试。这是手动完成的。

确保在生产集群中开启了 `query_log` 选项。

收集一天或更多的查询日志：

```
clickhouse-client --query="SELECT DISTINCT query FROM system.query_log WHERE event_date = today() AND query LIKE '%ym:%' AND query NOT LIKE '%system.query_log%' AND type = 2 AND is_initial_query" > queries.tsv
```

这是一个复杂的例子。`type = 2` 将过滤成功执行的查询。`query LIKE '%ym : %'` 用于从 Yandex.Metrica 中选择相关查询。`is_initial_query` 是仅选择由客户端发起的查询，而不是由 ClickHouse 本身（作为分布式查询处理的一部分）。

`scp` 这份日志到测试机器，并运行以下操作：

```
clickhouse benchmark --concurrency 16 < queries.tsv
```

(可能你需要指定运行的用户 `--user`)

然后离开它一晚或周末休息一下。

你要检查下 `clickhouse-server` 是否崩溃，内存占用是否合理，性能也不会随着时间的推移而降低。

由于查询和环境的高度可变性，不会记录精确的查询执行时序并且不进行比较。

编译测试

构建测试允许检查构建在各种替代配置和某些外部系统上是否被破坏。测试位于 `ci` 目录。它们从 Docker，Vagrant 中的源代码运行构建，有时在 Docker 中运行 `qemu-user-static`。这些测试正在开发中，测试运行不是自动化的。

动机：

通常我们会在 ClickHouse 构建的单个版本上发布并运行所有测试。但是有一些未经过彻底测试的替代构建版本。例子：

- 在 FreeBSD 中的构建；
- 在 Debian 中使用系统包中的库进行构建；
- 使用库的共享链接构建；
- 在 AArch64 平台进行构建。

例如，使用系统包构建是不好的做法，因为我们无法保证系统具有的确切版本的软件包。但 Debian 维护者确实需要这样做。出于这个原因，我们至少必须支持这种构建。另一个例子：共享链接是一个常见的麻烦来源，但是对于一些爱好者来说需要它。

虽然我们无法对所有构建版本运行所有测试，但我们想要检查至少不会破坏各种构建变体。为此，我们使用构建测试。

测试协议兼容性

当我们扩展 ClickHouse 网络协议时，我们手动测试旧的 clickhouse-client 与新的 clickhouse-server 和新的 clickhouse-client 一起使用旧的 clickhouse-server (只需从相应的包中运行二进制文件)

来自编译器的提示

ClickHouse 主要的代码 (位于 dbms 目录中) 使用 `-Wall -Wextra -Werror` 构建，并带有一些其他已启用的警告。虽然没有为第三方库启用这些选项。

Clang 有更多有用的警告 - 您可以使用 `-Weverything` 查找它们并选择默认构建的东西。

对于生产构建，使用 `gcc` (它仍然生成比 `clang` 稍高效的代码)。对于开发来说，`clang` 通常更方便使用。您可以使用调试模式在自己的机器上构建 (以节省笔记本电脑的电量)，但请注意，由于更好的控制流程和过程分析，编译器使用 `-O3` 会生成更多警告。当使用 `clang` 构建时，使用 `libc++` 而不是 `libstdc++`，并且在使用调试模式构建时，使用调试版本的 `libc++`，它允许在运行时捕获更多错误。

消毒剂

地址消毒剂。

我们在每个提交的基础上在 ASan 下运行功能和集成测试。

(`ASan`).

我们在 Valgrind 过夜进行功能测试。这需要几个小时。目前在 `re2` 库中有一个已知的误报，请参阅 [文章](#)。

螺纹消毒剂。

我们在 TSan 下进行功能测试。ClickHouse 必须通过所有测试。在 TSan 下运行不是自动化的，只是偶尔执行。

记忆消毒剂。

目前我们不使用 MSan。

未定义的行为消毒剂。

我们仍然不会在每次提交的基础上使用 UBSan。有一些地方需要解决。

调试分 alloc。

您可以使用 `DEBUG_TCMALLOC` CMake 选项启用 `tcmalloc` 的调试版本。我们在每次提交的基础上使用调试分配器运行测试。

更多请参阅 `tests/instructions/sanitizers.txt`。

模糊测试

我们使用简单的模糊测试来生成随机SQL查询并检查服务器是否正常，使用 Address sanitizer 执行模糊测试。你可以在 `00746_sql_fuzzy.pl` 找到它。测试应连续进行 (过夜和更长时间)。

截至2018年12月，我们仍然不使用库代码的孤立模糊测试。

安全审计

Yandex Cloud 部门的人员从安全角度对 ClickHouse 功能进行了一些基本概述。

静态分析

我们偶尔使用静态分析。我们已经评估过 `clang-tidy`，`Coverity`，`cppcheck`，`PVS-Studio`，`tscancode`。您将在 `tests/instructions/` 目录中找到使用说明。你也可以阅读[俄文文章](#)。

如果您使用 `CLion` 作为 IDE，您可以开箱即用一些 `clang-tidy` 检查。

其他强化

默认情况下使用 `FORTIFY_SOURCE`。它几乎没用，但在极少数情况下仍然有意义，我们不会禁用它。

代码风格

代码风格在[这里](#) 有说明。

要检查一些常见的样式冲突，您可以使用 `utils/check-style` 脚本。

为了强制你的代码的正确风格，你可以使用 `clang-format` 文件。`.clang-format` 位于源代码根目录，它主要与我们的实际代码风格对应。但不建议将 `clang-format` 应用于现有文件，因为它会使格式变得更糟。您可以使用 `clang-format-diff` 工具，您可以在 `clang` 源代码库中找到

或者，您可以尝试 `uncrustify` 工具来格式化您的代码。配置文件在源代码的根目录中的 `uncrustify.cfg`。它比 `clang-format` 经过更少的测试。

`CLion` 有自己的代码格式化程序，必须调整为我们的代码风格。

Metrica B2B 测试

每个 ClickHouse 版本都经过 Yandex Metrica 和 AppMetrica 引擎的测试。测试和稳定版本的 ClickHouse 部署在虚拟机上，并使用处理输入数据固定样本的度量引擎的小副本运行。将度量引擎的两个实例的结果一起进行比较

这些测试是由单独的团队自动完成的。由于移动部件的数量很多，大部分时间的测试都是完全无关的，很难弄清楚。很可能这些测试对我们来说是负值。然而，这些测试被证明是有用的大约一个或两个倍的数百。

测试覆盖率

截至2018年7月，我们不会跟踪测试复盖率。

自动化测试

我们使用 Yandex 内部 CI 和名为«沙箱»的作业自动化系统运行测试。我们还继续使用 Jenkins（可在Yandex内部使用）。

构建作业和测试在沙箱中按每次提交的基础上运行。结果包和测试结果发布在 GitHub 上，可以通过直接链接下载，结果会被永久存储。当您在 GitHub 上发送拉取请求时，我们将其标记为«可以测试»，我们的 CI 系统将为您构建 ClickHouse 包（发布，调试，地址消除等）。

由于时间和计算能力的限制，我们不使用 Travis CI。

在 Jenkins，我们运行字典测试，指标B2B测试。我们使用 Jenkins 来准备和发布版本。Jenkins是一种传统的技术，所有的工作将被转移到沙箱中。

ClickHose支持Linux,FreeBSD 及 Mac OS X 系统。

Windows 使用指引

如果您的系统是Windows，则需要创建Ubuntu虚拟机。可以安装VirtualBox来构建虚拟机。Ubuntu的下载链接为：<https://www.ubuntu.com/#download>。请使用下载好的镜像创建一个虚拟机（请确保虚拟机有至少4GB的内存容量）。在Ubuntu中使用«terminal»程序（gnome-terminal，konsole等）运行命令行终端，或使用快捷键Ctrl+Alt+T。

在 GitHub 上创建源码库

您需要(申请)一个GitHub账户来使用ClickHouse。

如果没有账户，请在<https://github.com>上注册一个。如果没有SSH密钥，请在本地创建密钥并将公钥上传到GitHub上。这有助于你提交更新代码。并且在不同的SSH服务端，你也可以使用相同的SSH密钥。

要创建ClickHouse源码库的分支，请在<https://github.com/ClickHouse/ClickHouse>页面上点击右上角的«fork»按钮。它会在本账户上创建您个人的ClickHouse/ClickHouse分支。

若要参与开发，首先请在ClickHouse的分支中提交您期望的变更，然后创建一个«pull请求»，以便这些变更能够被(ClickHouse/ClickHouse)主库接受。

请先安装git来使用git源码库。

请在Ubuntu终端上使用下列的指令来安装git:

```
sudo apt update  
sudo apt install git
```

在<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>中找到有关使用Git的简易手册。有关Git的详细手册，请参见: <https://git-scm.com/book/ru/v2>。

拷贝源码库到开发机

接下来，请将源码下载到开发机上。这步操作被称为«拷贝源码库»，是因为它在您的开发机上创建了源码库的本地副本。

在终端命令行输入下列指令：

```
git clone --recursive git@github.com:your_github_username/ClickHouse.git  
cd ClickHouse
```

请注意，您需要将`your_github_username`替换成实际使用的账户名！

这个指令将创建一个包含项目副本的ClickHouse工作目录。

重要的是，工作目录的路径中不应包含空格，因为这可能会导致运行构建系统时出现问题。

请注意，ClickHouse源码库使用了submodules。这是对其他库的引用（即项目所依赖的外部库）。即在拷贝源码库时，需要如上述指令中那样指定`--recursive`。如果在拷贝源码库时没有包含子模块，需要执行使用下列的指令：

```
git submodule init  
git submodule update
```

可以通过`git submodule status`来检查子模块的状态。

如果提示下列的错误信息：

```
Permission denied (publickey).  
fatal: Could not read from remote repository.
```

```
Please make sure you have the correct access rights  
and the repository exists.
```

这通常表示缺少用于连接GitHub的SSH密钥。这些密钥一般都在`~/.ssh`中。要接受SSH密钥，请在GitHub UI的设置页面中上传它们。

您还可以通过https协议来拷贝源码库：

```
git clone https://github.com/ClickHouse/ClickHouse.git
```

但是，这无法将变更提交到服务器上。您仍然可以暂时使用，并后续再添加SSH密钥，用`git remote`命令替换源码库的远程地址。

还可以将原始ClickHouse库的地址添加到本地库中，以便从那里获取更新：

```
git remote add upstream git@github.com:ClickHouse/ClickHouse.git
```

命令执行成功后，可以通过执行`git pull upstream master`，从ClickHouse的主分支中拉去更新。

使用子模块

在git中使用子模块可能会很痛苦。接下来的命令将有助于管理它：

```
# ! each command accepts --recursive
# Update remote URLs for submodules. Barely rare case
git submodule sync
# Add new submodules
git submodule init
# Update existing submodules to the current state
git submodule update
# Two last commands could be merged together
git submodule update --init
```

接下来的命令将帮助您将所有子模块重置为初始状态（！华林！ -里面的任何chenges将被删除）：

```
# Synchronizes submodules' remote URL with .gitmodules
git submodule sync --recursive
# Update the registered submodules with initialize not yet initialized
git submodule update --init --recursive
# Reset all changes done after HEAD
git submodule foreach git reset --hard
# Clean files from .gitignore
git submodule foreach git clean -xfd
# Repeat last 4 commands for all submodule
git submodule foreach git submodule sync --recursive
git submodule foreach git submodule update --init --recursive
git submodule foreach git submodule foreach git reset --hard
git submodule foreach git submodule foreach git clean -xfd
```

构建系统

ClickHouse使用 CMake 和 Ninja 来构建系统。

CMake - 一个可以生成Ninja文件的元构建系统（构建任务）。

Ninja - 一个轻量级的构建系统，专注于速度，用于执行这些cmake生成的任务。

在Ubuntu,Debian或者Mint系统上执行`sudo apt install cmake ninja-build`来安装ninja。

在CentOS,RedHat系统上执行`sudo yum install cmake ninja-build`。

如果您曾经使用过Arch或Gentoo，那么也许知道如何安装CMake。

若要在Mac OS X上安装CMake和Ninja，请先安装Homebrew，然后再通过brew安装其他内容：

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"  
brew install cmake ninja
```

接下来，检查CMake的版本：`cmake --version`。如果版本低于3.3，则需要从以下网站安装更新版本：<https://cmake.org/download/>。

可供选择的外部库

ClickHouse使用多个外部库进行构建。大多数外部库不需要单独安装，而是和ClickHouse一起在子模块中构建。可以查看[contrib](#)中罗列的清单。

C++ 编译器

GCC编译器从版本9开始，以及Clang版本 ≥ 8 都可支持构建ClickHouse。

Yandex官方当前使用GCC构建ClickHouse，因为它生成的机器代码性能较好（根据测评，最多可以相差几个百分点）。Clang通常可以更加便捷的开发。我们的持续集成（CI）平台会运行大约十二种构建组合的检查。

在Ubuntu上安装GCC，请执行：`sudo apt install gcc g++`

请使用`gcc --version`查看gcc的版本。如果gcc版本低于9，请参考此处的指示：<https://clickhouse.tech/docs/en/development/build/#install-gcc-9>。

在Mac OS X上安装GCC，请执行：`brew install gcc`

如果您决定使用Clang，还可以同时安装`libc++`以及`lld`，前提是您也熟悉它们。此外，也推荐使用`ccache`。

构建的过程

如果当前已经准备好构建ClickHouse，我们建议您在ClickHouse中创建一个单独的目录`build`，其中包含所有构建组件：

```
mkdir build  
cd build
```

您也可以有多个不同类型的构建目录（例如，`build_release`, `build_debug`等等）。

在`build`目录下，通过运行CMake配置构建。在第一次运行之前，请定义用于指定编译器的环境变量（本示例中为gcc 9编译器）。

```
export CC=gcc-9 CXX=g++-9  
cmake ..
```

`CC`变量指代C的编译器（C Compiler的缩写），而`CXX`变量指代要使用哪个C++编译器进行编译。

为了更快的构建，请使用`debug`构建类型-不含优化的构建。为此提供以下的参数`-D CMAKE_BUILD_TYPE=Debug`:

```
cmake -D CMAKE_BUILD_TYPE=Debug ..
```

您可以通过在`build`目录中运行此命令来更改构建类型。

运行ninja进行构建:

```
ninja clickhouse-server clickhouse-client
```

在此示例中，仅将构建所需的二进制文件。

如果您需要构建所有的二进制文件（utilities和tests），请运行不带参数的ninja：

```
ninja
```

全量构建需要大约30GB的可用磁盘空间或15GB的空间来构建主要的二进制文件。

当构建的机器上有大量内存时，可考虑设置与-j参数并行运行的构建任务数量：

```
ninja -j 1 clickhouse-server clickhouse-client
```

在拥有4GB内存的机器上，建议设置成1，在拥有8GB内存的机器上，建议按-j 2设置。

如果您收到以下消息：

```
ninja : error : loading'build.ninja' : No such file or directory
```

则表示生成构建配置失败，请检查上述消息。

成功启动构建过程后，您将看到构建进度-已处理任务的数量和任务总数。

在libhdfs2库中生成有关protobuf文件的消息时，可能会显示诸如 libprotobuf WARNING。它们没有影响，可以忽略不计。

成功构建后，会得到一个可执行文件 ClickHouse/<build_dir>/programs/clickhouse:

```
ls -l programs/clickhouse
```

运行ClickHouse可执行文件

要以当前的用户身份运行服务，请进入到 ClickHouse/programs/server/ 目录（在 build 文件夹外）并运行：

```
../../../../build/programs/clickhouse server
```

在这种情况下，ClickHouse将使用位于当前目录中的配置文件。您可以从任何目录运行 Clickhouse server，并将配置文件--config-file 的路径指定为命令行参数。

在另外一个终端上连接ClickHouse的clickhouse-client客户端，请进入到 ClickHouse/build/programs/ 并运行 clickhouse client。

如果您在Mac OS X 或者 FreeBSD上收到 Connection refused 的消息，请尝试指定主机地址为127.0.0.1：

```
clickhouse client --host 127.0.0.1
```

您可以使用自定义构建的ClickHouse二进制文件替换系统中安装的ClickHouse二进制文件的生成版本。为此，请参照官方网站上的说明在计算机上安装ClickHouse。接下来，运行以下命令：

```
sudo service clickhouse-server stop  
sudo cp ClickHouse/build/programs/clickhouse /usr/bin/  
sudo service clickhouse-server start
```

请注意，clickhouse-client，clickhouse-server和其他服务通常共享 clickhouse 二进制文件的符号链接。

您还可以使用系统上安装的ClickHouse软件包中的配置文件运行自定义构建的ClickHouse二进制文件：

```
sudo service clickhouse-server stop  
sudo -u clickhouse ClickHouse/build/programs/clickhouse server --config_file /etc/clickhouse_server/config.xml
```

```
sudo -u clickhouse clickhouse/build/bin/programs/clickhouse-server --config-file /etc/clickhouse-server/config.xml
```

IDE (集成开发环境)

如果您还不知道使用哪款IDE，我们推荐使用CLion。CLion是一款商业软件，但能够有30天的免费使用时间。它同时也对学生免费。CLion可以在Linux和Mac OS X上使用。

KDevelop和QTCreator是另外两款适合开发ClickHouse的替代IDE。尽管不太稳定，但KDevelop还是作为一款非常便捷的IDE。如果KDevelop在打开项目后不久崩溃，则您应该在打开项目文件列表后立即单击《全部停止》按钮。按此处理后，KDevelop可以正常使用。

作为简易的代码编辑器，您可以使用Sublime Text或Visual Studio Code或Kate（在Linux上都可用）。

值得一提的是CLion会创建自己的build路径，它还会自行选择debug作为构建类型。对于配置，它使用CLion中定义的CMake版本，而不是您安装的版本。最后，CLion会使用make而不是ninja去构建任务。这属于正常的现象，请记住这一点，以免造成混淆。

编写代码

ClickHouse的架构描述可以在此处查看：<https://clickhouse.tech/docs/en/development/architecture/>

代码风格指引：<https://clickhouse.tech/docs/en/development/style/>

编写测试用例：<https://clickhouse.tech/docs/en/development/tests/>

任务列表：https://github.com/ClickHouse/ClickHouse/blob/master/tests/instructions/easy_tasks_sorted_en.md

测试数据

开发ClickHouse通常需要加载现实的数据集，尤其是在性能测试的场景。我们可以从Yandex.Metrica获取一组特别准备的匿名数据。这些数据需要额外使用3GB的空闲磁盘空间。请注意，完成大多数开发任务并不需要此数据。

```
sudo apt install wget xz-utils

wget https://clickhouse-datasets.s3.yandex.net/hits/tsv/hits_v1.tsv.xz
wget https://clickhouse-datasets.s3.yandex.net/visits/tsv/visits_v1.tsv.xz

xz -v -d hits_v1.tsv.xz
xz -v -d visits_v1.tsv.xz

clickhouse-client

CREATE TABLE test.hits ( WatchID UInt64, JavaEnable UInt8, Title String, GoodEvent Int16, EventTime DateTime,
EventDate Date, CounterID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RegionID UInt32, UserID UInt64,
CounterClass Int8, OS UInt8, UserAgent UInt8, URL String, Referer String, URLDomain String, RefererDomain String,
Refresh UInt8, IsRobot UInt8, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions
Array(UInt32), RefererRegions Array(UInt32), ResolutionWidth UInt16, ResolutionHeight UInt16, ResolutionDepth
UInt8, FlashMajor UInt8, FlashMinor UInt8, FlashMinor2 String, NetMajor UInt8, NetMinor UInt8, UserAgentMajor
UInt16, UserAgentMinor FixedString(2), CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, MobilePhone
UInt8, MobilePhoneModel String, Params String, IPNetworkID UInt32, TraficSourceID Int8, SearchEngineID UInt16,
SearchPhrase String, AdvEngineID UInt8, IsArtifical UInt8, WindowClientWidth UInt16, WindowClientHeight UInt16,
ClientTimeZone Int16, ClientEventTime DateTime, SilverlightVersion1 UInt8, SilverlightVersion2 UInt8,
SilverlightVersion3 UInt32, SilverlightVersion4 UInt16, PageCharset String, CodeVersion UInt32, IsLink UInt8,
IsDownload UInt8, IsNotBounce UInt8, FUniqID UInt64, HID UInt32, IsOldCounter UInt8, IsEvent UInt8, IsParameter
UInt8, DontCountHits UInt8, WithHash UInt8, HitColor FixedString(1), UTCEventTime DateTime, Age UInt8, Sex UInt8,
Income UInt8, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), RemoteIP UInt32, RemoteIP6
FixedString(16), WindowName Int32, OpenerName Int32, HistoryLength Int16, BrowserLanguage FixedString(2),
BrowserCountry FixedString(2), SocialNetwork String, SocialAction String, HTTPError UInt16, SendTiming Int32,
DNSTiming Int32, ConnectTiming Int32, ResponseStartTiming Int32, ResponseEndTiming Int32, FetchTiming Int32,
RedirectTiming Int32, DOMInteractiveTiming Int32, DOMContentLoadedTiming Int32, DOMCompleteTiming Int32,
LoadEventStartTiming Int32, LoadEventEndTiming Int32, NSTDOMContentLoadedTiming Int32, FirstPaintTiming Int32,
```

```

RedirectCount Int8, SocialSourceNetworkID UInt8, SocialSourcePage String, ParamPrice Int64, ParamOrderID String,
ParamCurrency FixedString(3), ParamCurrencyID UInt16, GoalsReached Array(UInt32), OpenstatServiceName String,
OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String,
UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, RefererHash UInt64,
URLHash UInt64, CLID UInt32, YCLID UInt64, ShareService String, ShareURL String, ShareTitle String,
`ParsedParams.Key1` Array(String), `ParsedParams.Key2` Array(String), `ParsedParams.Key3` Array(String),
`ParsedParams.Key4` Array(String), `ParsedParams.Key5` Array(String), `ParsedParams.ValueDouble` Array(Float64),
IslandID FixedString(16), RequestNum UInt32, RequestTry UInt8) ENGINE = MergeTree PARTITION BY
toYYYYMM(EventDate) SAMPLE BY intHash32(UserID) ORDER BY (CounterID, EventDate, intHash32(UserID), EventTime);

```

```

CREATE TABLE test.visits ( CounterID UInt32, StartDate Date, Sign Int8, IsNew UInt8, VisitID UInt64, UserID UInt64,
StartTime DateTime, Duration UInt32, UTCStartTime DateTime, PageViews Int32, Hits Int32, IsBounce UInt8, Referer
String, StartURL String, RefererDomain String, StartURLDomain String, EndURL String, LinkURL String, IsDownload
UInt8, TraficSourceID Int8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8, PlaceID Int32,
RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions
Array(UInt32), IsYandex UInt8, GoalReachesDepth Int32, GoalReachesURL Int32, GoalReachesAny Int32,
SocialSourceNetworkID UInt8, SocialSourcePage String, MobilePhoneModel String, ClientEventTime DateTime,
RegionID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RemoteIP UInt32, RemoteIP6 FixedString(16), IPNetworkID
UInt32, SilverlightVersion3 UInt32, CodeVersion UInt32, ResolutionWidth UInt16, ResolutionHeight UInt16,
UserAgentMajor UInt16, UserAgentMinor UInt16, WindowClientWidth UInt16, WindowClientHeight UInt16,
SilverlightVersion2 UInt8, SilverlightVersion4 UInt16, FlashVersion3 UInt16, FlashVersion4 UInt16, ClientTimeZone
Int16, OS UInt8, UserAgent UInt8, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, NetMajor UInt8,
NetMinor UInt8, MobilePhone UInt8, SilverlightVersion1 UInt8, Age UInt8, Sex UInt8, Income UInt8, JavaEnable UInt8,
CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, BrowserLanguage UInt16, BrowserCountry UInt16,
Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), Params Array(String), `Goals.ID` Array(UInt32),
`Goals.Serial` Array(UInt32), `Goals.EventTime` Array(DateTime), `Goals.Price` Array(Int64), `Goals.OrderID` Array(String),
`Goals.CurrencyID` Array(UInt32), WatchIDs Array(UInt64), ParamSumPrice Int64, ParamCurrency
FixedString(3), ParamCurrencyID UInt16, ClickLogID UInt64, ClickEventID Int32, ClickGoodEvent Int32, ClickEventTime
DateTime, ClickPriorityID Int32, ClickPhraseID Int32, ClickPageID Int32, ClickPlaceID Int32, ClickTypeID Int32,
ClickResourceID Int32, ClickCost UInt32, ClickClientIP UInt32, ClickDomainID UInt32, ClickURL String, ClickAttempt
UInt8, ClickOrderID UInt32, ClickBannerID UInt32, ClickMarketCategoryID UInt32, ClickMarketPP UInt32,
ClickMarketCategoryName String, ClickMarketPPName String, ClickAWAPSCampaignName String, ClickPageName
String, ClickTargetType UInt16, ClickTargetPhraseID UInt64, ClickContextType UInt8, ClickSelectType Int8,
ClickOptions String, ClickGroupBannerID Int32, OpenstatServiceName String, OpenstatCampaignID String,
OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String, UTMCampaign String,
UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, FirstVisit DateTime, PredLastVisit Date,
LastVisit Date, TotalVisits UInt32, `TraficSource.ID` Array(Int8), `TraficSource.SearchEngineID` Array(UInt16),
`TraficSource.AdvEngineID` Array(UInt8), `TraficSource.PlaceID` Array(UInt16), `TraficSource.SocialSourceNetworkID` Array(UInt8),
`TraficSource.Domain` Array(String), `TraficSource.SearchPhrase` Array(String),
`TraficSource.SocialSourcePage` Array(String), Attendance FixedString(16), CLID UInt32, YCLID UInt64,
NormalizedRefererHash UInt64, SearchPhraseHash UInt64, RefererDomainHash UInt64, NormalizedStartURLHash
UInt64, StartURLDomainHash UInt64, NormalizedEndURLHash UInt64, TopLevelDomain UInt64, URLscheme UInt64,
OpenstatServiceNameHash UInt64, OpenstatCampaignIDHash UInt64, OpenstatAdIDHash UInt64,
OpenstatSourceIDHash UInt64, UTMSourceHash UInt64, UTMMediumHash UInt64, UTMCampaignHash UInt64,
UTMContentHash UInt64, UTMTermHash UInt64, FromHash UInt64, WebVisorEnabled UInt8, WebVisorActivity UInt32,
`ParsedParams.Key1` Array(String), `ParsedParams.Key2` Array(String), `ParsedParams.Key3` Array(String),
`ParsedParams.Key4` Array(String), `ParsedParams.Key5` Array(String), `ParsedParams.ValueDouble` Array(Float64),
`Market.Type` Array(UInt8), `Market.GoalID` Array(UInt32), `Market.OrderID` Array(String), `Market.OrderPrice` Array(Int64),
`Market.PP` Array(UInt32), `Market.DirectPlaceID` Array(UInt32), `Market.DirectOrderID` Array(UInt32),
`Market.DirectBannerID` Array(UInt32), `Market.GoodID` Array(String), `Market.GoodName` Array(String),
`Market.GoodQuantity` Array(Int32), `Market.GoodPrice` Array(Int64), IslandID FixedString(16)) ENGINE =
CollapsingMergeTree(Sign) PARTITION BY toYYYYMM(StartDate) SAMPLE BY intHash32(UserID) ORDER BY (CounterID,
StartDate, intHash32(UserID), VisitID);

```

```

clickhouse-client --max_insert_block_size 100000 --query "INSERT INTO test.hits FORMAT TSV" < hits_v1.tsv
clickhouse-client --max_insert_block_size 100000 --query "INSERT INTO test.visits FORMAT TSV" < visits_v1.tsv

```

创建拉取请求

进入到GitHub 用户界面中的fork库。如果您已经在某个分支中进行开发，则需要选择该分支。在屏幕中有一个《拉取请求书...》按钮，点击它并从下拉菜单中选择“从分支”。

水》的按钮。头标上遵守了《创建一个项目以接受对仓库的变更》。

即使工作尚未完成，也可以创建拉取请求。在这种情况下，请在标题的开头加上«WIP»（正在进行中），以便后续更改。这对于协同审查和讨论更改以及运行所有可用测试用例很有用。提供有关变更的简短描述很重要，这将在后续用于生成重新发布变更日志。

Yandex成员一旦在您的拉取请求上贴上«可以测试»标签，就会开始测试。一些初始检查项（例如，代码类型）的结果会在几分钟内反馈。构建的检查结果将在半小时内完成。而主要的测试用例集结果将在一小时内报告给您。

系统将分别为您的拉取请求准备ClickHouse二进制版本。若要检索这些构建信息，请在检查列表中单击« ClickHouse构建检查»旁边的«详细信息»链接。在这里，您会找到指向ClickHouse的.deb软件包的直接链接，此外，甚至可以将其部署在生产服务器上（如果您不担心）。

某些构建项很可能会在首次构建时失败。这是因为我们同时检查了基于gcc和clang的构建，几乎所有现有的被clang启用的警告（总是带有-Werror标志）。在同一页面上，您可以找到所有构建的日志，因此不必以所有可能的方式构建ClickHouse。

使用的三方库

图书馆	许可
base64	BSD2-条款许可
升压	提升软件许可证1.0
brotli	MIT
capnproto	MIT
cctz	Apache许可证2.0
双转换	BSD3-条款许可
FastMemcpy	MIT
googletest	BSD3-条款许可
超扫描	BSD3-条款许可
libbtrie	BSD2-条款许可
libcxxabi	BSD + MIT
libdivide	Zlib许可证
libgsasl	LGPL v2.1
libhdfs3	Apache许可证2.0
libmetrohash	Apache许可证2.0
libpcg-随机	Apache许可证2.0
libressl	OpenSSL许可证

libatk	BSD2-条款许可
libwidechar_width	CC0 1.0通用
llvm	BSD3-条款许可
lz4	BSD2-条款许可
mariadb-连接器-c	LGPL v2.1
murmurhash	公共领域
pdqsort	Zlib许可证
poco	提升软件许可证-1.0版
protobuf	BSD3-条款许可
re2	BSD3-条款许可
UnixODBC	LGPL v2.1
zlib-ng	Zlib许可证
zstd	BSD3-条款许可
## 在 Mac OS X 中编译 ClickHouse	

ClickHouse 支持在 Mac OS X 10.12 版本中编译。若您在用更早的操作系统版本，可以尝试在指令中使用 [Gentoo Prefix](#) 和 `clang sl`。

通过适当的更改，它应该可以适用于任何其他的 Linux 发行版。

安装 Homebrew

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

安装编译器，工具库

```
brew install cmake ninja gcc icu4c mariadb-connector-c openssl libtool gettext
```

拉取 ClickHouse 源码

```
git clone --recursive git@github.com:ClickHouse/ClickHouse.git
## or: git clone --recursive https://github.com/ClickHouse/ClickHouse.git
cd ClickHouse
```

编译 ClickHouse

```
mkdir build
cd build
cmake .. -DCMAKE_CXX_COMPILER=`which g++-8` -DCMAKE_C_COMPILER=`which gcc-8`
```

```
ninja  
cd ..
```

注意事项

若你想运行 clickhouse-server，请先确保增加系统的最大文件数配置。

注意

可能需要用 sudo

为此，请创建以下文件：

/图书馆/LaunchDaemons/限制.plist

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"  
      "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
  <key>Label</key>  
  <string>limit.maxfiles</string>  
  <key>ProgramArguments</key>  
  <array>  
    <string>launchctl</string>  
    <string>limit</string>  
    <string>maxfiles</string>  
    <string>524288</string>  
    <string>524288</string>  
  </array>  
  <key>RunAtLoad</key>  
  <true/>  
  <key>ServiceIPC</key>  
  <false/>  
</dict>  
</plist>
```

执行以下命令：

```
$ sudo chown root:wheel /Library/LaunchDaemons/limit.plist
```

然后重启。

可以通过 `ulimit -n` 命令来检查是否生效。

如何在Linux中编译Mac OS X ClickHouse

Linux机器也可以编译运行在OS X系统的clickhouse二进制包，这可以用于在Linux上跑持续集成测试。如果要在Mac OS X上直接构建ClickHouse，请参考另外一篇指南：https://clickhouse.tech/docs/zh/development/build_osx/

Mac OS X的交叉编译基于以下构建说明，请首先遵循它们。

安装Clang-8

按照<https://apt.llvm.org/>中的说明进行Ubuntu或Debian安装。

例如，安装Bionic的命令如下：

```
sudo echo "deb [trusted=yes] http://apt.llvm.org/bionic/ llvm-toolchain-bionic-8 main" >> /etc/apt/sources.list
sudo apt-get install clang-8
```

安装交叉编译工具集

我们假设安装 cctools 在 \${CCTOOLS} 路径下

```
mkdir ${CCTOOLS}

git clone https://github.com/tpoechtrager/apple-libtapi.git
cd apple-libtapi
INSTALLPREFIX=${CCTOOLS} ./build.sh
./install.sh
cd ..

git clone https://github.com/tpoechtrager/cctools-port.git
cd cctools-port/cctools
./configure --prefix=${CCTOOLS} --with-libtapi=${CCTOOLS} --target=x86_64-apple-darwin
make install

cd ${CCTOOLS}
wget https://github.com/phracker/MacOSX-SDKs/releases/download/10.14-beta4/MacOSX10.14.sdk.tar.xz
tar xJf MacOSX10.14.sdk.tar.xz
```

编译 ClickHouse

```
cd ClickHouse
mkdir build-osx
CC=clang-8 CXX=clang++-8 cmake . -Bbuild-osx -DCMAKE_SYSTEM_NAME=Darwin \
-DCMAKE_AR:FILEPATH=${CCTOOLS}/bin/x86_64-apple-darwin-ar \
-DCMAKE_RANLIB:FILEPATH=${CCTOOLS}/bin/x86_64-apple-darwin-ranlib \
-DLINKER_NAME=${CCTOOLS}/bin/x86_64-apple-darwin-ld \
-DSDK_PATH=${CCTOOLS}/MacOSX10.14.sdk
ninja -C build-osx
```

生成的二进制文件将具有Mach-O可执行格式，并且不能在Linux上运行。

如何构建 ClickHouse 发布包

安装 Git 和 Pbuilder

```
sudo apt-get update
sudo apt-get install git pbuilder debhelper lsb-release fakeroot sudo debian-archive-keyring debian-keyring
```

拉取 ClickHouse 源码

```
git clone --recursive https://github.com/ClickHouse/ClickHouse.git
cd ClickHouse
```

运行发布脚本

```
./release
```

如何在开发过程中编译 ClickHouse

以下教程是在 Ubuntu Linux 中进行编译的示例。

通过适当的更改，它应该可以适用于任何其他的 Linux 发行版。

仅支持具有 x86_64、AArch64。对 Power9 的支持是实验性的。

安装 Git 和 CMake 和 Ninja

```
sudo apt-get install git cmake ninja-build
```

或cmake3而不是旧系统上的cmake。

或者在早期版本的系统中用 cmake3 替代 cmake

安装 GCC 9

有几种方法可以做到这一点。

安装 PPA 包

```
sudo apt-get install software-properties-common  
sudo apt-add-repository ppa:ubuntu-toolchain-r/test  
sudo apt-get update  
sudo apt-get install gcc-9 g++-9
```

源码安装 gcc

请查看 [utils/ci/build-gcc-from-sources.sh](#)

使用 GCC 9 来编译

```
export CC=gcc-9  
export CXX=g++-9
```

拉取 ClickHouse 源码

```
git clone --recursive git@github.com:ClickHouse/ClickHouse.git  
## or: git clone --recursive https://github.com/ClickHouse/ClickHouse.git  
  
cd ClickHouse
```

编译 ClickHouse

```
mkdir build  
cd build  
cmake ..  
ninja  
cd ..
```

若要创建一个执行文件，执行 `ninja clickhouse`。

这个命令会使得 `programs(clickhouse)` 文件可执行，您可以使用 `client` 或 `server` 参数运行。

如何编写 C++ 代码

一般建议

1. 以下是建议，而不是要求。

- 如果你在修改代码，遵守已有的风格是有意义的。
- 代码的风格需保持一致。一致的风格有利于阅读代码，并且方便检索代码。
- 许多规则没有逻辑原因；它们是由既定的做法决定的。

格式化

- 大多数格式化可以用 `clang-format` 自动完成。
- 缩进是4个空格。配置开发环境，使得 `TAB` 代表添加四个空格。
- 左右花括号需在单独的行。

```
inline void readBoolText(bool & x, ReadBuffer & buf)
{
    char tmp = '0';
    readChar(tmp, buf);
    x = tmp != '0';
}
```

- 若整个方法体仅有一行 `描述`，则可以放到单独的行上。在花括号周围放置空格（除了行尾的空格）。

```
inline size_t mask() const { return buf_size() - 1; }
inline size_t place(HashValue x) const { return x & mask(); }
```

- 对于函数。不要在括号周围放置空格。

```
void reinsert(const Value & x)
```

```
memcpy(&buf[place_value], &x, sizeof(x));
```

- 在 `if`，`for`，`while`和其他表达式中，在开括号前面插入一个空格（与函数声明相反）。

```
for (size_t i = 0; i < rows; i += storage.index_granularity)
```

- 在二元运算符（`+`，`-`，`*`，`/`，`%`，...）和三元运算符 `?:` 周围添加空格。

```
UInt16 year = (s[0] - '0') * 1000 + (s[1] - '0') * 100 + (s[2] - '0') * 10 + (s[3] - '0');
UInt8 month = (s[5] - '0') * 10 + (s[6] - '0');
UInt8 day = (s[8] - '0') * 10 + (s[9] - '0');
```

- 若有换行，新行应该以运算符开头，并且增加对应的缩进。

```
if (elapsed_ns)
    message << "("
        << rows_read_on_server * 1000000000 / elapsed_ns << " rows/s., "
        << bytes_read_on_server * 1000.0 / elapsed_ns << " MB/s.) ";
```

- 如果需要，可以在一行内使用空格来对齐。

```
dst.ClickLogID      = click.LogID;
dst.ClickEventID    = click.EventID;
```

```
dst.ClickGoodEvent = click.GoodEvent;
```

10. 不要在`.`，`->`周围加入空格

如有必要，运算符可以包裹到下一行。在这种情况下，它前面的偏移量增加。

11. 不要使用空格来分开一元运算符`(--, ++, *, &, ...)`和参数。

12. 在逗号后面加一个空格，而不是在之前。同样的规则也适合`for`循环中的分号。

13. 不要用空格分开`[]`运算符。

14. 在`template <...>`表达式中，在`template`和`<`中加入一个空格，在`<`后面或在`>`前面都不要有空格。

```
template <typename TKey, typename TValue>
struct AggregatedStatElement
{}
```

15. 在类和结构体中，`public`，`private`以及`protected`同`class/struct`无需缩进，其他代码须缩进。

```
template <typename T>
class MultiVersion
{
public:
    /// Version of object for usage. shared_ptr manage lifetime of version.
    using Version = std::shared_ptr<const T>;
    ...
}
```

16. 如果对整个文件使用相同的`namespace`，并且没有其他重要的东西，则`namespace`中不需要偏移量。

17. 在`if`，`for`，`while`中包裹的代码块中，若代码是一个单行的`statement`，那么大括号是可选的。可以将`statement`放到一行中。这个规则同样适用于嵌套的`if`，`for`，`while`，...

但是如果内部`statement`包含大括号或`else`，则外部块应该用大括号括起来。

```
/// Finish write.
for (auto & stream : streams)
    stream.second->finalize();
```

18. 行的某尾不应该包含空格。

19. 源文件应该用`UTF-8`编码。

20. 非ASCII字符可用于字符串文字。

```
<< ", " << (timer.elapsed() / chunks_stats.hits) << " μsec(hit);"
```

21. 不要在一行中写入多个表达式。

22. 将函数内部的代码段分组，并将它们与不超过一行的空行分开。

23. 将`函数`，`类`用一个或两个空行分开。

24. `const`必须写在类型名称之前。

```
//correct
```

```
//incorrect
const char * pos
const std::string & s
//incorrect
char const * pos
```

25. 声明指针或引用时，`*` 和 `&` 符号两边应该都用空格分隔。

```
//correct
const char * pos
//incorrect
const char* pos
const char *pos
```

26. 使用模板类型时，使用 `using` 关键字对它们进行别名（最简单的情况除外）。

换句话说，模板参数仅在 `using` 中指定，并且不在代码中重复。

`using` 可以在本地声明，例如在函数内部。

```
//correct
using FileStreams = std::map<std::string, std::shared_ptr<Stream>>;
FileStreams streams;
//incorrect
std::map<std::string, std::shared_ptr<Stream>> streams;
```

27. 不要在一个语句中声明不同类型的多个变量。

```
//incorrect
int x, *y;
```

28. 不要使用C风格的类型转换。

```
//incorrect
std::cerr << (int)c << std::endl;
//correct
std::cerr << static_cast<int>(c) << std::endl;
```

29. 在类和结构中，组成员和函数分别在每个可见范围内。

30. 对于小类和结构，没有必要将方法声明与实现分开。

对于任何类或结构中的小方法也是如此。

对于模板化类和结构，不要将方法声明与实现分开（因为否则它们必须在同一个转换单元中定义）

31. 您可以将换行规则定在140个字符，而不是80个字符。

32. 如果不需要 `postfix`，请始终使用前缀增量/减量运算符。

```
for (Names::const_iterator it = column_names.begin(); it != column_names.end(); ++it)
```

评论

1. 请务必为所有非常重要的代码部分添加注释。

这是非常重要的。编写注释可能会帮助您意识到代码不是必需的，或者设计错误。

```
/** Part of piece of memory, that can be used.  
 * For example, if internal_buffer is 1MB, and there was only 10 bytes loaded to buffer from file for reading,  
 * then working_buffer will have size of only 10 bytes  
 * (working_buffer.end() will point to position right after those 10 bytes available for read).  
 */
```

2. 注释可以尽可能详细。

3. 在他们描述的代码之前放置注释。在极少数情况下，注释可以在代码之后，在同一行上。

```
/** Parses and executes the query.  
 */  
void executeQuery(  
    ReadBuffer & istr, /// Where to read the query from (and data for INSERT, if applicable)  
    WriteBuffer & ostr, /// Where to write the result  
    Context & context, /// DB, tables, data types, engines, functions, aggregate functions...  
    BlockInputStreamPtr & query_plan, /// Here could be written the description on how query was executed  
    QueryProcessingStage::Enum stage = QueryProcessingStage::Complete /// Up to which stage process the SELECT  
    query  
)
```

4. 注释应该只用英文撰写。

5. 如果您正在编写库，请在主头文件中包含解释它的详细注释。

6. 请勿添加无效的注释。特别是，不要留下像这样的空注释：

```
/*  
 * Procedure Name:  
 * Original procedure name:  
 * Author:  
 * Date of creation:  
 * Dates of modification:  
 * Modification authors:  
 * Original file name:  
 * Purpose:  
 * Intent:  
 * Designation:  
 * Classes used:  
 * Constants:  
 * Local variables:  
 * Parameters:  
 * Date of creation:  
 * Purpose:  
 */
```

这个示例来源于 <http://home.tamk.fi/~jaalto/course/coding-style/doc/unmaintainable-code/>。

7. 不要在每个文件的开头写入垃圾注释（作者，创建日期...）。

8. 单行注释用三个斜杆：///，多行注释以 /** 开始。这些注释会当做文档。

注意：您可以使用 Doxygen 从这些注释中生成文档。但是通常不使用 Doxygen，因为在 IDE 中导航代码更方便。

9. 多行注释的开头和结尾不得有空行（关闭多行注释的行除外）。

10. 要注释掉代码，请使用基本注释，而不是《记录》注释。

11. 在提交之前删除代码的无效注释部分。

12. 不要在注释或代码中使用亵渎语言。

13. 不要使用大写字母。不要使用过多的标点符号。

```
/// WHAT THE FAIL???
```

14. 不要使用注释来制作分隔符。

```
////////////////////////////////////////////////////////////////////////
```

15. 不要在注释中开始讨论。

```
/// Why did you do this stuff?
```

16. 没有必要在块的末尾写一条注释来描述它的含义。

```
/// for
```

姓名

1. 在变量和类成员的名称中使用带下划线的小写字母。

```
size_t max_block_size;
```

2. 对于函数（方法）的名称，请使用以小写字母开头的驼峰标识。

```
std::string getName() const override { return "Memory"; }
```

3. 对于类（结构）的名称，使用以大写字母开头的驼峰标识。接口名称用 I 前缀。

```
class StorageMemory : public IStorage
```

4. `using` 的命名方式与类相同，或者以 `_t`` 命名。

5. 模板类型参数的名称：在简单的情况下，使用 `T; T`，`U; T1, T2`。

对于更复杂的情况，要么遵循类名规则，要么添加前缀 `T`。

```
template <typename TKey, typename TValue>
struct AggregatedStatElement
```

6. 模板常量参数的名称：遵循变量名称的规则，或者在简单的情况下使用 `N`。

```
template <bool without_www>
struct ExtractDomain
```

7. 对于抽象类型（接口），用 `I` 前缀。

```
class IBlockInputStream
```

8. 如果在本地使用变量，则可以使用短名称。

在所有其他情况下，请使用能描述含义的名称。

```
bool info_successfully_loaded = false;
```

9. `define` 和全局常量的名称使用带下划线的 ALL_CAPS。

```
##define MAX_SRC_TABLE_NAMES_TO_STORE 1000
```

10. 文件名应使用与其内容相同的样式。

如果文件包含单个类，则以与该类名称相同的方式命名该文件。

如果文件包含单个函数，则以与函数名称相同的方式命名文件。

11. 如果名称包含缩写，则：

- 对于变量名，缩写应使用小写字母 `mysql_connection`（不是 `mySQL_connection`）。
- 对于类和函数的名称，请将大写字母保留在缩写 `MySQLConnection`（不是 `MySqlConnection`）。

12. 仅用于初始化类成员的构造方法参数的命名方式应与类成员相同，但最后使用下划线。

```
FileQueueProcessor(
    const std::string & path_,
    const std::string & prefix_,
    std::shared_ptr<FileHandler> handler_
) : path(path_),
    prefix(prefix_),
    handler(handler_),
    log(&Logger::get("FileQueueProcessor"))
{}
```

如果构造函数体中未使用该参数，则可以省略下划线后缀。

13. 局部变量和类成员的名称没有区别（不需要前缀）。

```
timer (not m_timer)
```

14. 对于 `enum` 中的常量，请使用带大写字母的驼峰标识。ALL_CAPS 也可以接受。如果 `enum` 是非本地的，请使用 `enum class`。

```
enum class CompressionMethod
{
    QuickLZ = 0,
    LZ4     = 1,
};
```

15. 所有名字必须是英文。不允许音译俄语单词。

```
not Stroka
```

16. 缩写须是众所周知的（当您可以在维基百科或搜索引擎中轻松找到缩写的含义时）。

```
'AST', `SQL`.  
Not `NVDH` (some random letters)
```

如果缩短版本是常用的，则可以接受不完整的单词。

如果注释中旁边包含全名，您也可以使用缩写。

17. C++ 源码文件名称必须为 `.cpp` 拓展名。头文件必须为 `.h` 拓展名。

如何编写代码

1. 内存管理。

手动内存释放 (`delete`) 只能在库代码中使用。

在库代码中，`delete` 运算符只能在析构函数中使用。

在应用程序代码中，内存必须由拥有它的对象释放。

示例：

- 最简单的方法是将对象放在堆栈上，或使其成为另一个类的成员。
- 对于大量小对象，请使用容器。
- 对于自动释放少量在堆中的对象，可以用 `shared_ptr/unique_ptr`。

2. 资源管理。

使用 `RAII` 以及查看以上说明。

3. 错误处理。

在大多数情况下，您只需要抛出一个异常，而不需要捕获它（因为 `RAII`）。

在离线数据处理应用程序中，通常可以接受不捕获异常。

在处理用户请求的服务器中，通常足以捕获连接处理程序顶层的异常。

在线程函数中，你应该在 `join` 之后捕获并保留所有异常以在主线程中重新抛出它们。

```
/// If there weren't any calculations yet, calculate the first block synchronously  
if (!started)  
{  
    calculate();  
    started = true;  
}  
else // If calculations are already in progress, wait for the result  
    pool.wait();  
  
if (exception)  
    exception->rethrow();
```

不处理就不要隐藏异常。永远不要盲目地把所有异常都记录到日志中。

```
//Not correct  
catch (...) {}
```

```
catch (const DB::Exception & e)
{
    if (e.code() == ErrorCodes::UNKNOWN_AGGREGATE_FUNCTION)
        return nullptr;
    else
        throw;
}
```

当使用具有返回码或 `errno` 的函数时，请始终检查结果并在出现错误时抛出异常。

```
if (0 != close(fd))
    throwFromErrno("Cannot close file " + file_name, ErrorCodes::CANNOT_CLOSE_FILE);
```

不要使用断言。

4. 异常类型。

不需要在应用程序代码中使用复杂的异常层次结构。系统管理员应该可以理解异常文本。

5. 从析构函数中抛出异常。

不建议这样做，但允许这样做。

按照以下选项：

- 创建一个函数（`done()` 或 `finalize()`），它将提前完成所有可能导致异常的工作。如果调用了该函数，则稍后在析构函数中应该没有异常。
- 过于复杂的任务（例如通过网络发送消息）可以放在单独的方法中，类用户必须在销毁之前调用它们。
- 如果析构函数中存在异常，则最好记录它而不是隐藏它（如果 `logger` 可用）。
- 在简单的应用程序中，依赖于 `std::terminate`（对于 C++ 11 中默认情况下为 `noexcept` 的情况）来处理异常是可以接受的。

6. 匿名代码块。

您可以在单个函数内创建单独的代码块，以使某些变量成为局部变量，以便在退出块时调用析构函数。

```
Block block = data.in->read();

{
    std::lock_guard<std::mutex> lock(mutex);
    data.ready = true;
    data.block = block;
}

ready_any.set();
```

7. 多线程。

在离线数据处理程序中：

- 尝试在单个CPU核心上获得最佳性能。然后，您可以根据需要并行化代码。

在服务端应用中：

- 使用线程池来处理请求。此时，我们还没有任何需要用户空间上下文切换的任务。

Fork 不用于并行化。

8. 同步线程。

通常可以使不同的线程使用不同的存储单元（甚至更好：不同的缓存线），并且不使用任何线程同步（除了 `joinAll`）。

如果需要同步，在大多数情况下，在 `lock_guard` 下使用互斥量就足够了。

在其他情况下，使用系统同步原语。不要使用忙等待。

仅在最简单的情况下才应使用原子操作。

除非是您的主要专业领域，否则不要尝试实施无锁数据结构。

9. 指针和引用。

大部分情况下，请用引用。

10. 常量。

使用 `const` 引用，指向常量的指针，`const_iterator` 和 `const` 指针。

将 `const` 视为默认值，仅在必要时使用非 `const`。

当按值传递变量时，使用 `const` 通常没有意义。

11. 无符号。

必要时使用 `unsigned`。

12. 数值类型。

使用 `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, 以及 `Int64`, `size_t`, `ssize_t` 还有 `ptrdiff_t`。

不要使用这些类型：`signed / unsigned long`, `long long`, `short`, `signed / unsigned char`, `char`。

13. 参数传递。

通过引用传递复杂类型（包括 `std::string`）。

如果函数中传递堆中创建的对象，则使参数类型为 `shared_ptr` 或者 `unique_ptr`。

14. 返回值

大部分情况下使用 `return`。不要使用 `[return std::move(res)]{ .strike }`。

如果函数在堆上分配对象并返回它，请使用 `shared_ptr` 或 `unique_ptr`。

在极少数情况下，您可能需要通过参数返回值。在这种情况下，参数应该是引用传递的。

```
using AggregateFunctionPtr = std::shared_ptr<IAggregateFunction>;\n\n/** Allows creating an aggregate function by its name.\n */\nclass AggregateFunctionFactory\n{\npublic:\n    AggregateFunctionFactory();\n    AggregateFunctionPtr get(const String & name, const DataTypes & argument_types) const;
```

15. 命名空间。

没有必要为应用程序代码使用单独的 `namespace`。

小型库也不需要这个。

对于中大型库，须将所有代码放在 `namespace` 中。

在库的 `.h` 文件中，您可以使用 `namespace detail` 来隐藏应用程序代码不需要的实现细节。

在 `.cpp` 文件中，您可以使用 `static` 或匿名命名空间来隐藏符号。

同样 `namespace` 可用于 `enum` 以防止相应的名称落入外部 `namespace`（但最好使用 `enum class`）。

16. 延迟初始化。

如果初始化需要参数，那么通常不应该编写默认构造函数。

如果稍后您需要延迟初始化，则可以添加将创建无效对象的默认构造函数。或者，对于少量对象，您可以使用 `shared_ptr` / `unique_ptr`。

```
Loader(DB::Connection * connection_, const std::string & query, size_t max_block_size_);  
  
/// For deferred initialization  
Loader() {}
```

17. 虚函数。

如果该类不是用于多态使用，则不需要将函数设置为虚拟。这也适用于析构函数。

18. 编码。

在所有情况下使用 UTF-8 编码。使用 `std::string` 和 `char*`。不要使用 `std::wstring` 和 `wchar_t`。

19. 日志。

请参阅代码中的示例。

在提交之前，删除所有无意义和调试日志记录，以及任何其他类型的调试输出。

应该避免循环记录日志，即使在 Trace 级别也是如此。

日志必须在任何日志记录级别都可读。

在大多数情况下，只应在应用程序代码中使用日志记录。

日志消息必须用英文写成。

对于系统管理员来说，日志最好是可以理解的。

不要在日志中使用亵渎语言。

在日志中使用UTF-8编码。在极少数情况下，您可以在日志中使用非ASCII字符。

20. 输入-输出。

不要使用 `iostreams` 在对应用程序性能至关重要的内部循环中（并且永远不要使用 `stringstream`）。

使用 `DB/IO` 库替代。

21. 日期和时间。

参考 `DateLUT` 库。

22. 引入头文件。

一直用 `#pragma once` 而不是其他宏。

23. using 语法

`using namespace` 不会被使用。您可以使用特定的 `using`。但是在类或函数中使它成为局部的。

24. 不要使用 trailing return type 为必要的功能。

```
[auto f() -> void]{.strike}
```

25. 声明和初始化变量。

```
//right way  
std::string s = "Hello";  
std::string s{"Hello"};  
  
//wrong way  
auto s = std::string{"Hello"};
```

26. 对于虚函数，在基类中编写 `virtual`，但在后代类中写 `override` 而不是 `virtual`。

没有用到的 C++ 特性。

1. 不使用虚拟继承。

2. 不使用 C++03 中的异常标准。

平台

1. 我们为特定平台编写代码。

但在其他条件相同的情况下，首选跨平台或可移植代码。

2. 语言：C++17.

3. 编译器： `gcc`。此时（2017年12月），代码使用7.2版编译。（它也可以使用 `clang 4` 编译）

使用标准库 (`libstdc++` 或 `libc++`)。

4. 操作系统：Linux Ubuntu，不比 Precise 早。

5. 代码是为x86_64 CPU架构编写的。

CPU指令集是我们服务器中支持的最小集合。目前，它是SSE 4.2。

6. 使用 `-Wall -Wextra -Werror` 编译参数。

7. 对所有库使用静态链接，除了那些难以静态连接的库（参见 `ldd` 命令的输出）。

8. 使用发布的设置来开发和调试代码。

工具

1. KDevelop 是一个好的 IDE.

2. 调试可以使用 `gdb`， `valgrind (memcheck)`， `strace`， `-fsanitize=...`，或 `tcmalloc_minimal_debug`.

3. 对于性能分析，使用 `Linux Perf`， `valgrind (callgrind)`，或者 `strace -cf`。

4. 源代码用 `Git` 作版本控制。

5. 使用 `CMake` 构建。

6. 程序的发布使用 `deb` 安装包。

7. 提交到 master 分支的代码不能破坏编译。

虽然只有选定的修订被认为是可行的。

8. 尽可能经常地进行提交，即使代码只是部分准备好了。

目的明确的功能，使用分支。

如果 master 分支中的代码尚不可构建，请在 push 之前将其从构建中排除。您需要在几天内完成或删除它。

9. 对于不重要的更改，请使用分支并在服务器上发布它们。

10. 未使用的代码将从 repo 中删除。

库

1. 使用C++ 14标准库（允许实验性功能），以及 boost 和 Poco 框架。

2. 如有必要，您可以使用 OS 包中提供的任何已知库。

如果有一个好的解决方案已经可用，那就使用它，即使这意味着你必须安装另一个库。

（但要准备从代码中删除不好的库）

3. 如果软件包没有您需要的软件包或者有过时的版本或错误的编译类型，则可以安装不在软件包中的库。

4. 如果库很小并且没有自己的复杂构建系统，请将源文件放在 contrib 文件夹中。

5. 始终优先考虑已经使用的库。

一般建议

1. 尽可能精简代码。

2. 尝试用最简单的方式实现。

3. 在你知道代码是如何工作以及内部循环如何运作之前，不要编写代码。

4. 在最简单的情况下，使用 using 而不是类或结构。

5. 如果可能，不要编写复制构造函数，赋值运算符，析构函数（虚拟函数除外，如果类包含至少一个虚函数），移动构造函数或移动赋值运算符。换句话说，编译器生成的函数必须正常工作。您可以使用 default。

6. 鼓励简化代码。尽可能减小代码的大小。

其他建议

1. 从 stddef.h 明确指定 std:: 的类型。

不推荐。换句话说，我们建议写 size_t 而不是 std::size_t，因为它更短。

也接受添加 std::。

2. 为标准C库中的函数明确指定 std::

不推荐。换句话说，写 memcpy 而不是 std::memcpy。

原因是存在类似的非标准功能，例如 memmem。我们偶尔会使用这些功能。namespace std 中不存在这些函数。

如果你到处都写 std::memcpy 而不是 memcpy，那么没有 std:: 的 memmem 会显得很奇怪。

不过，如果您愿意，仍然可以使用 std::。

3. 当标准C++库中提供相同的函数时，使用C中的函数。

如果它更高效，这是可以接受的。

例如，使用 `memcpy` 而不是 `std::copy` 来复制大块内存。

4. 函数的多行参数。

允许以下任何包装样式：

```
function(  
    T1 x1,  
    T2 x2)
```

```
function(  
    size_t left, size_t right,  
    const & RangesInDataParts ranges,  
    size_t limit)
```

```
function(size_t left, size_t right,  
        const & RangesInDataParts ranges,  
        size_t limit)
```

```
function(size_t left, size_t right,  
        const & RangesInDataParts ranges,  
        size_t limit)
```

```
function(  
    size_t left,  
    size_t right,  
    const & RangesInDataParts ranges,  
    size_t limit)
```

点击教程

从本教程中可以期待什么？

通过本教程，您将学习如何设置一个简单的ClickHouse集群。它会很小，但容错和可扩展。然后，我们将使用其中一个示例数据集来填充数据并执行一些演示查询。

单节点设置

为了推迟分布式环境的复杂性，我们将首先在单个服务器或虚拟机上部署ClickHouse。ClickHouse通常是从安装 黑布 或 `rpm` 包，但也有 替代办法 对于不支持它们的操作系统。

例如，您选择了 `deb` 包和执行：

```
sudo apt-get install apt-transport-https ca-certificates dirmngr  
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv E0C56BD4  
  
echo "deb https://repo.clickhouse.tech/deb/stable/ main/" | sudo tee \  
/etc/apt/sources.list.d/clickhouse.list  
sudo apt-get update
```

```
sudo apt-get install -y clickhouse-server clickhouse-client
```

```
sudo service clickhouse-server start  
clickhouse-client
```

我们在安装的软件包中有什么：

- `clickhouse-client` 包包含 “环板client” 应用程序，交互式ClickHouse控制台客户端。
- `clickhouse-common` 包包含一个ClickHouse可执行文件。
- `clickhouse-server` 包包含要作为服务器运行ClickHouse的配置文件。

服务器配置文件位于 `/etc/clickhouse-server/`. 在进一步讨论之前，请注意 `<path>` 元素在 `config.xml`. `Path` 确定数据存储的位置，因此应该位于磁盘容量较大的卷上；默认值为 `/var/lib/clickhouse/`. 如果你想调整配置，直接编辑并不方便 `config.xml` 文件，考虑到它可能会在未来的软件包更新中被重写。复盖配置元素的推荐方法是创建 在配置文件。d 目录 它作为“patches”要配置。`xml`

你可能已经注意到了，`clickhouse-server` 安装包后不会自动启动。它也不会在更新后自动重新启动。您启动服务器的方式取决于您的init系统，通常情况下，它是：

```
sudo service clickhouse-server start
```

或

```
sudo /etc/init.d/clickhouse-server start
```

服务器日志的默认位置是 `/var/log/clickhouse-server/`. 服务器已准备好处理客户端连接一旦它记录 `Ready for connections` 消息

一旦 `clickhouse-server` 正在运行我们可以利用 `clickhouse-client` 连接到服务器并运行一些测试查询，如 `SELECT "Hello, world!"`.

► Clickhouse-客户端的快速提示

导入示例数据集

现在是时候用一些示例数据填充我们的ClickHouse服务器。在本教程中，我们将使用Yandex的匿名数据。Metrica，在成为开源之前以生产方式运行ClickHouse的第一个服务（更多关于这一点 [历史科](#)）. 有 [多种导入Yandex的方式](#)。梅里卡数据集，为了本教程，我们将使用最现实的一个。

下载并提取表数据

```
curl https://clickhouse-datasets.s3.yandex.net/hits/tsv/hits_v1.tsv.xz | unxz --threads=`nproc` > hits_v1.tsv  
curl https://clickhouse-datasets.s3.yandex.net/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
```

提取的文件大小约为 10GB。

创建表

与大多数数据库管理系统一样，ClickHouse在逻辑上将表分组为“databases”. 有一个 `default` 数据库，但我们将创建一个名为新的 `tutorial`:

```
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS tutorial"
```

与数据库相比，创建表的语法要复杂得多（请参阅 [参考资料](#). 一般 `CREATE TABLE` 声明必须指定三个关键的事情：

- 要创建的表的名称。
- Table schema, i.e. list of columns and their **数据类型**。
- 表引擎** 及其设置，这决定了如何物理执行对此表的查询的所有细节。

YandexMetrica是一个网络分析服务，样本数据集不包括其全部功能，因此只有两个表可以创建：

- `hits` 是一个表格，其中包含所有用户在服务所涵盖的所有网站上完成的每个操作。
- `visits` 是一个包含预先构建的会话而不是单个操作的表。

让我们看看并执行这些表的实际创建表查询：

```
CREATE TABLE tutorial.hits_v1
(
    `WatchID` UInt64,
    `JavaEnable` UInt8,
    `Title` String,
    `GoodEvent` Int16,
    `EventTime` DateTime,
    `EventDate` Date,
    `CounterID` UInt32,
    `ClientIP` UInt32,
    `ClientIP6` FixedString(16),
    `RegionID` UInt32,
    `UserID` UInt64,
    `CounterClass` Int8,
    `OS` UInt8,
    `UserAgent` UInt8,
    `URL` String,
    `Referer` String,
    `URLDomain` String,
    `RefererDomain` String,
    `Refresh` UInt8,
    `IsRobot` UInt8,
    `RefererCategories` Array(UInt16),
    `URLCategories` Array(UInt16),
    `URLRegions` Array(UInt32),
    `RefererRegions` Array(UInt32),
    `ResolutionWidth` UInt16,
    `ResolutionHeight` UInt16,
    `ResolutionDepth` UInt8,
    `FlashMajor` UInt8,
    `FlashMinor` UInt8,
    `FlashMinor2` String,
    `NetMajor` UInt8,
    `NetMinor` UInt8,
    `UserAgentMajor` UInt16,
    `UserAgentMinor` FixedString(2),
    `CookieEnable` UInt8,
    `JavascriptEnable` UInt8,
    `IsMobile` UInt8,
    `MobilePhone` UInt8,
    `MobilePhoneModel` String,
    `Params` String,
    `IPNetworkID` UInt32,
    `TraficSourceID` Int8,
    `SearchEngineID` UInt16,
    `SearchPhrase` String,
    `AdvEngineID` UInt8,
    `IsArtifical` UInt8,
    `WindowClientWidth` UInt16,
    `WindowClientHeight` UInt16,
    `ClientTimezone` Int16
)
```

```
ClientTimezone` UInt64,  
`ClientEventTime` DateTime,  
`SilverlightVersion1` UInt8,  
`SilverlightVersion2` UInt8,  
`SilverlightVersion3` UInt32,  
`SilverlightVersion4` UInt16,  
`PageCharset` String,  
`CodeVersion` UInt32,  
`IsLink` UInt8,  
`IsDownload` UInt8,  
` IsNotBounce` UInt8,  
`FUniqID` UInt64,  
`HID` UInt32,  
`IsOldCounter` UInt8,  
`IsEvent` UInt8,  
`IsParameter` UInt8,  
`DontCountHits` UInt8,  
`WithHash` UInt8,  
`HitColor` FixedString(1),  
`UTCEventTime` DateTime,  
`Age` UInt8,  
`Sex` UInt8,  
`Income` UInt8,  
`Interests` UInt16,  
`Robotness` UInt8,  
`GeneralInterests` Array(UInt16),  
`RemoteIP` UInt32,  
`RemoteIP6` FixedString(16),  
`WindowName` Int32,  
`OpenerName` Int32,  
`HistoryLength` Int16,  
`BrowserLanguage` FixedString(2),  
`BrowserCountry` FixedString(2),  
`SocialNetwork` String,  
`SocialAction` String,  
`HTTPError` UInt16,  
`SendTiming` Int32,  
`DNSTiming` Int32,  
`ConnectTiming` Int32,  
`ResponseStartTiming` Int32,  
`ResponseEndTiming` Int32,  
`FetchTiming` Int32,  
`RedirectTiming` Int32,  
`DOMInteractiveTiming` Int32,  
`DOMContentLoadedTiming` Int32,  
`DOMCompleteTiming` Int32,  
`LoadEventStartTiming` Int32,  
`LoadEventEndTiming` Int32,  
`NSToDOMContentLoadedTiming` Int32,  
`FirstPaintTiming` Int32,  
`RedirectCount` Int8,  
`SocialSourceNetworkID` UInt8,  
`SocialSourcePage` String,  
`ParamPrice` Int64,  
`ParamOrderID` String,  
`ParamCurrency` FixedString(3),  
`ParamCurrencyID` UInt16,  
`GoalsReached` Array(UInt32),  
`OpenstatServiceName` String,  
`OpenstatCampaignID` String,  
`OpenstatAdID` String,  
`OpenstatSourceID` String,  
`UTMSource` String.
```

```

    `UTMSource` String,
    `UTMMedium` String,
    `UTMCampaign` String,
    `UTMContent` String,
    `UTMTerm` String,
    `FromTag` String,
    `HasGCLID` UInt8,
    `RefererHash` UInt64,
    `URLHash` UInt64,
    `CLID` UInt32,
    `YCLID` UInt64,
    `ShareService` String,
    `ShareURL` String,
    `ShareTitle` String,
    `ParsedParams` Nested(
        Key1 String,
        Key2 String,
        Key3 String,
        Key4 String,
        Key5 String,
        ValueDouble Float64),
    `IslandID` FixedString(16),
    `RequestNum` UInt32,
    `RequestTry` UInt8
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID)
SETTINGS index_granularity = 8192

```

```

CREATE TABLE tutorial.visits_v1
(
    `CounterID` UInt32,
    `StartDate` Date,
    `Sign` Int8,
    `IsNew` UInt8,
    `VisitID` UInt64,
    `UserID` UInt64,
    `StartTime` DateTime,
    `Duration` UInt32,
    `UTCStartTime` DateTime,
    `PageViews` Int32,
    `Hits` Int32,
    `IsBounce` UInt8,
    `Referer` String,
    `StartURL` String,
    `RefererDomain` String,
    `StartURLDomain` String,
    `EndURL` String,
    `LinkURL` String,
    `IsDownload` UInt8,
    `TraficSourceID` Int8,
    `SearchEngineID` UInt16,
    `SearchPhrase` String,
    `AdvEngineID` UInt8,
    `PlaceID` Int32,
    `RefererCategories` Array(UInt16),
    `URLCategories` Array(UInt16),
    `URLRegions` Array(UInt32),
    `RefererRegions` Array(UInt32),
    `IsYandex` UInt8
)

```

```
  `Index` Int32,
  `GoalReachesDepth` Int32,
  `GoalReachesURL` Int32,
  `GoalReachesAny` Int32,
  `SocialSourceNetworkID` UInt8,
  `SocialSourcePage` String,
  `MobilePhoneModel` String,
  `ClientEventTime` DateTime,
  `RegionID` UInt32,
  `ClientIP` UInt32,
  `ClientIP6` FixedString(16),
  `RemoteIP` UInt32,
  `RemoteIP6` FixedString(16),
  `IPNetworkID` UInt32,
  `SilverlightVersion3` UInt32,
  `CodeVersion` UInt32,
  `ResolutionWidth` UInt16,
  `ResolutionHeight` UInt16,
  `UserAgentMajor` UInt16,
  `UserAgentMinor` UInt16,
  `WindowClientWidth` UInt16,
  `WindowClientHeight` UInt16,
  `SilverlightVersion2` UInt8,
  `SilverlightVersion4` UInt16,
  `FlashVersion3` UInt16,
  `FlashVersion4` UInt16,
  `ClientTimeZone` Int16,
  `OS` UInt8,
  `UserAgent` UInt8,
  `ResolutionDepth` UInt8,
  `FlashMajor` UInt8,
  `FlashMinor` UInt8,
  `NetMajor` UInt8,
  `NetMinor` UInt8,
  `MobilePhone` UInt8,
  `SilverlightVersion1` UInt8,
  `Age` UInt8,
  `Sex` UInt8,
  `Income` UInt8,
  `JavaEnable` UInt8,
  `CookieEnable` UInt8,
  `JavascriptEnable` UInt8,
  `IsMobile` UInt8,
  `BrowserLanguage` UInt16,
  `BrowserCountry` UInt16,
  `Interests` UInt16,
  `Robotness` UInt8,
  `GeneralInterests` Array(UInt16),
  `Params` Array(String),
  `Goals` Nested(
    ID UInt32,
    Serial UInt32,
    EventTime DateTime,
    Price Int64,
    OrderID String,
    CurrencyID UInt32),
  `WatchIDs` Array(UInt64),
  `ParamSumPrice` Int64,
  `ParamCurrency` FixedString(3),
  `ParamCurrencyID` UInt16,
  `ClickLogID` UInt64,
  `ClickEventID` Int32,
  `ClickGoodEvent` Int32,
```

```
`ClickEventTime` DateTime,  
`ClickPriorityID` Int32,  
`ClickPhraseID` Int32,  
`ClickPageID` Int32,  
`ClickPlaceID` Int32,  
`ClickTypeID` Int32,  
`ClickResourceID` Int32,  
`ClickCost` UInt32,  
`ClickClientIP` UInt32,  
`ClickDomainID` UInt32,  
`ClickURL` String,  
`ClickAttempt` UInt8,  
`ClickOrderID` UInt32,  
`ClickBannerID` UInt32,  
`ClickMarketCategoryID` UInt32,  
`ClickMarketPP` UInt32,  
`ClickMarketCategoryName` String,  
`ClickMarketPPName` String,  
`ClickAWAPSCampaignName` String,  
`ClickPageName` String,  
`ClickTargetType` UInt16,  
`ClickTargetPhraseID` UInt64,  
`ClickContextType` UInt8,  
`ClickSelectType` Int8,  
`ClickOptions` String,  
`ClickGroupBannerID` Int32,  
`OpenstatServiceName` String,  
`OpenstatCampaignID` String,  
`OpenstatAdID` String,  
`OpenstatSourceID` String,  
`UTMSource` String,  
`UTMMedium` String,  
`UTMCampaign` String,  
`UTMContent` String,  
`UTMTerm` String,  
`FromTag` String,  
`HasGCLID` UInt8,  
`FirstVisit` DateTime,  
`PredLastVisit` Date,  
`LastVisit` Date,  
`TotalVisits` UInt32,  
`TraficSource` Nested(  
    ID Int8,  
    SearchEnginID UInt16,  
    AdvEnginID UInt8,  
    PlaceID UInt16,  
    SocialSourceNetworkID UInt8,  
    Domain String,  
    SearchPhrase String,  
    SocialSourcePage String),  
`Attendance` FixedString(16),  
`CLID` UInt32,  
`YCLID` UInt64,  
`NormalizedRefererHash` UInt64,  
`SearchPhraseHash` UInt64,  
`RefererDomainHash` UInt64,  
`NormalizedStartURLHash` UInt64,  
`StartURLDomainHash` UInt64,  
`NormalizedEndURLHash` UInt64,  
`TopLevelDomain` UInt64,  
`URLScheme` UInt64,  
`OpenstatServiceNameHash` UInt64,
```

```

`OpenstatCampaignIDHash` UInt64,
`OpenstatAdIDHash` UInt64,
`OpenstatSourceIDHash` UInt64,
`UTMSourceHash` UInt64,
`UTMMediumHash` UInt64,
`UTMCampaignHash` UInt64,
`UTMContentHash` UInt64,
`UTMTermHash` UInt64,
`FromHash` UInt64,
`WebVisorEnabled` UInt8,
`WebVisorActivity` UInt32,
`ParsedParams` Nested(
    Key1 String,
    Key2 String,
    Key3 String,
    Key4 String,
    Key5 String,
    ValueDouble Float64),
`Market` Nested(
    Type UInt8,
    GoalID UInt32,
    OrderID String,
    OrderPrice Int64,
    PP UInt32,
    DirectPlaceID UInt32,
    DirectOrderID UInt32,
    DirectBannerID UInt32,
    GoodID String,
    GoodName String,
    GoodQuantity Int32,
    GoodPrice Int64),
`IslandID` FixedString(16)
)
ENGINE = CollapsingMergeTree(Sign)
PARTITION BY toYYYYMM(StartDate)
ORDER BY (CounterID, StartDate, intHash32(UserID), VisitID)
SAMPLE BY intHash32(UserID)
SETTINGS index_granularity = 8192

```

您可以使用以下交互模式执行这些查询 `clickhouse-client`（只需在终端中启动它，而不需要提前指定查询）或尝试一些 [替代接口](#) 如果你愿意的话

正如我们所看到的，`hits_v1` 使用 [基本MergeTree引擎](#)，而 `visits_v1` 使用 [崩溃变体](#)。

导入数据

数据导入到ClickHouse是通过以下方式完成的 [INSERT INTO](#) 查询像许多其他SQL数据库。然而，数据通常是在一个提供[支持的序列化格式](#) 而不是 `VALUES` 子句（也支持）。

我们之前下载的文件是以制表符分隔的格式，所以这里是如何通过控制台客户端导入它们：

```

clickhouse-client --query "INSERT INTO tutorial.hits_v1 FORMAT TSV" --max_insert_block_size=100000 < hits_v1.tsv
clickhouse-client --query "INSERT INTO tutorial.visits_v1 FORMAT TSV" --max_insert_block_size=100000 < visits_v1.tsv

```

ClickHouse有很多 [要调整的设置](#) 在控制台客户端中指定它们的一种方法是通过参数，我们可以看到 `--max_insert_block_size`。找出可用的设置，它们意味着什么以及默认值的最简单方法是查询 `system.settings` 表：

```

SELECT name, value, changed, description
FROM system.settings
WHERE name LIKE '%max_insert_b%'

```

FORMAT TSV

```
max_insert_block_size 1048576 0 "The maximum block size for insertion, if we control the creation of blocks for insertion."
```

您也可以 **OPTIMIZE** 导入后的表。使用MergeTree-family引擎配置的表总是在后台合并数据部分以优化数据存储（或至少检查是否有意义）。这些查询强制表引擎立即进行存储优化，而不是稍后进行一段时间：

```
clickhouse-client --query "OPTIMIZE TABLE tutorial.hits_v1 FINAL"
clickhouse-client --query "OPTIMIZE TABLE tutorial.visits_v1 FINAL"
```

这些查询开始一个I/O和CPU密集型操作，所以如果表一直接收到新数据，最好不要管它，让合并在后台运行。

现在我们可以检查表导入是否成功：

```
clickhouse-client --query "SELECT COUNT(*) FROM tutorial.hits_v1"
clickhouse-client --query "SELECT COUNT(*) FROM tutorial.visits_v1"
```

查询示例

```
SELECT
    StartURL AS URL,
    AVG(Duration) AS AvgDuration
FROM tutorial.visits_v1
WHERE StartDate BETWEEN '2014-03-23' AND '2014-03-30'
GROUP BY URL
ORDER BY AvgDuration DESC
LIMIT 10
```

```
SELECT
    sum(Sign) AS visits,
    sumIf(Sign, has(Goals.ID, 1105530)) AS goal_visits,
    (100. * goal_visits) / visits AS goal_percent
FROM tutorial.visits_v1
WHERE (CounterID = 912887) AND (toYYYYMM(StartDate) = 201403) AND (domain(StartURL) = 'yandex.ru')
```

集群部署

ClickHouse集群是一个同质集群。设置步骤：

1. 在群集的所有计算机上安装ClickHouse服务器
2. 在配置文件中设置群集配置
3. 在每个实例上创建本地表
4. 创建一个 **分布式表**

分布式表 实际上是一种“view”到ClickHouse集群的本地表。从分布式表中选择查询使用集群所有分片的资源执行。您可以为多个集群指定**configs**，并创建多个分布式表，为不同的集群提供视图。

具有三个分片的集群的示例配置，每个分片一个副本：

```
<remote_servers>
<perftest_3shards_1replicas>
  <shard>
    <replica>
      <host>example-perftest01j.yandex.ru</host>
      <ports>9000</ports>
```

```
<port>9000</port>
</replica>
</shard>
<shard>
<replica>
<host>example-perftest02j.yandex.ru</host>
<port>9000</port>
</replica>
</shard>
<shard>
<replica>
<host>example-perftest03j.yandex.ru</host>
<port>9000</port>
</replica>
</shard>
</perftest_3shards_1replicas>
</remote_servers>
```

为了进一步演示，让我们创建一个新的本地表 `CREATE TABLE` 我们用于查询 `hits_v1`，但不同的表名：

```
CREATE TABLE tutorial.hits_local (...) ENGINE = MergeTree() ...
```

创建提供集群本地表视图的分布式表：

```
CREATE TABLE tutorial.hits_all AS tutorial.hits_local
ENGINE = Distributed(perftest_3shards_1replicas, tutorial, hits_local, rand());
```

常见的做法是在集群的所有计算机上创建类似的分布式表。它允许在群集的任何计算机上运行分布式查询。还有一个替代选项可以使用以下方法为给定的`SELECT`查询创建临时分布式表 **远程** 表功能。

我们走吧 **INSERT SELECT** 将该表传播到多个服务器。

```
INSERT INTO tutorial.hits_all SELECT * FROM tutorial.hits_v1;
```

碌莽禄Notice:

这种方法不适合大型表的分片。有一个单独的工具 **ツ環板-ヨツ嘉ツツ惣** 这可以重新分片任意大表。

正如您所期望的那样，如果计算量大的查询使用3台服务器而不是一个，则运行速度快N倍。

在这种情况下，我们使用了具有3个分片的集群，每个分片都包含一个副本。

为了在生产环境中提供弹性，我们建议每个分片应包含分布在多个可用区或数据中心（或至少机架）之间的2-3个副本。请注意，ClickHouse支持无限数量的副本。

包含三个副本的一个分片集群的示例配置：

```
<remote_servers>
...
<perftest_1shards_3replicas>
<shard>
<replica>
<host>example-perftest01j.yandex.ru</host>
<port>9000</port>
</replica>
<replica>
```

```
<host>example-perf02j.yandex.ru</host>
<port>9000</port>
</replica>
<replica>
<host>example-perf03j.yandex.ru</host>
<port>9000</port>
</replica>
</shard>
</perf02_1shards_3replicas>
</remote_servers>
```

启用本机复制 [动物园管理员](#) 是必需的。ClickHouse负责所有副本的数据一致性，并在失败后自动运行恢复过程。建议将ZooKeeper集群部署在单独的服务器上（其中没有其他进程，包括ClickHouse正在运行）。

注

ZooKeeper不是一个严格的要求：在某些简单的情况下，您可以通过将数据写入应用程序代码中的所有副本 来复制数据。这种方法是不建议，在这种情况下，ClickHouse将无法保证所有副本上的数据一致性。因此，它成为您的应用程序的责任。

ZooKeeper位置在配置文件中指定：

```
<zookeeper>
<node>
<host>zoo01.yandex.ru</host>
<port>2181</port>
</node>
<node>
<host>zoo02.yandex.ru</host>
<port>2181</port>
</node>
<node>
<host>zoo03.yandex.ru</host>
<port>2181</port>
</node>
</zookeeper>
```

此外，我们需要设置宏来识别每个用于创建表的分片和副本：

```
<macros>
<shard>01</shard>
<replica>01</replica>
</macros>
```

如果在创建复制表时没有副本，则会实例化新的第一个副本。如果已有实时副本，则新副本将克隆现有副本中的数据。您可以选择首先创建所有复制的表，然后向其中插入数据。另一种选择是创建一些副本，并在数据插入之后或期间添加其他副本。

```
CREATE TABLE tutorial.hits_replica (...)  
ENGINE = ReplicatedMergeTree(  
    '/clickhouse_perf02j/tables/{shard}/hits',  
    '{replica}'  
)  
...
```

在这里，我们使用 [ReplicatedMergeTree](#) 引擎。在参数中，我们指定句令分片和副本标识符的ZooKeeper路径。

```
INSERT INTO tutorial.hits_replica SELECT * FROM tutorial.hits_local;
```

复制在多主机模式下运行。数据可以加载到任何副本中，然后系统会自动将其与其他实例同步。复制是异步的，因此在给定时刻，并非所有副本都可能包含最近插入的数据。至少应有一个副本允许数据摄取。其他人将同步数据和修复一致性，一旦他们将再次变得活跃。请注意，这种方法允许最近插入的数据丢失的可能性很低。

示例数据集

本节介绍如何获取示例数据集并将其导入ClickHouse。

对于某些数据集示例查询也可用。

- 脱敏的Yandex.Metrica数据集
- 星型基准测试
- 维基访问数据
- Criteo TB级别点击日志
- AMPLab大数据基准测试
- 纽约出租车数据
- 航班飞行数据

脱敏的Yandex.Metrica数据集

Dataset由两个表组成，其中包含有关命中中的匿名数据 (`hits_v1`) 和访问 (`visits_v1`) 的Yandex的。梅特里卡 你可以阅读更多关于Yandex的。梅特里卡 [ClickHouse历史](#) 科。

数据集由两个表组成，其中任何一个都可以作为压缩表下载 `tsv.xz` 文件或作为准备的分区。除此之外，该扩展版本 `hits` 包含1亿行的表可作为TSV在https://clickhouse-datasets.s3.yandex.net/hits/tsv/hits_100m_obfuscated_v1.tsv.xz 并作为准备的分区在https://clickhouse-datasets.s3.yandex.net/hits/partitions/hits_100m_obfuscated_v1.tar.xz。

从准备好的分区获取表

下载和导入点击表:

```
curl -O https://clickhouse-datasets.s3.yandex.net/hits/partitions/hits_v1.tar
tar xvf hits_v1.tar -C /var/lib/clickhouse # path to ClickHouse data directory
## check permissions on unpacked data, fix if required
sudo service clickhouse-server restart
clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1"
```

下载和导入访问:

```
curl -O https://clickhouse-datasets.s3.yandex.net/visits/partitions/visits_v1.tar
tar xvf visits_v1.tar -C /var/lib/clickhouse # path to ClickHouse data directory
## check permissions on unpacked data, fix if required
sudo service clickhouse-server restart
clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1"
```

从压缩TSV文件获取表

从压缩的TSV文件下载并导入命中:

https://clickhouse-datasets.s3.yandex.net/hits/partitions/hits_100m_obfuscated_v1.tar.xz

```

curl https://clickhouse-datasets.s3.yandex.net/hits/tsv/hits_v1.tsv.xz | unxz --threads= nproc > hits_v1.tsv
## now create table
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
clickhouse-client --query "CREATE TABLE datasets.hits_v1 ( WatchID UInt64, JavaEnable UInt8, Title String, GoodEvent Int16, EventTime DateTime, EventDate Date, CounterID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RegionID UInt32, UserID UInt64, CounterClass UInt8, OS UInt8, UserAgent UInt8, URL String, Referer String, URLDomain String, RefererDomain String, Refresh UInt8, IsRobot UInt8, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions Array(UInt32), ResolutionWidth UInt16, ResolutionHeight UInt16, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, FlashMinor2 String, NetMajor UInt8, NetMinor UInt8, UserAgentMajor UInt16, UserAgentMinor FixedString(2), CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, MobilePhone UInt8, MobilePhoneModel String, Params String, IPNetworkID UInt32, TraficSourceID UInt8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8, IsArtifical UInt8, WindowClientWidth UInt16, WindowClientHeight UInt16, ClientTimeZone Int16, ClientEventTime DateTime, SilverlightVersion1 UInt8, SilverlightVersion2 UInt8, SilverlightVersion3 UInt32, SilverlightVersion4 UInt16, PageCharset String, CodeVersion UInt32, IsLink UInt8, IsDownload UInt8, IsNotBounce UInt8, FUniqID UInt64, HID UInt32, IsOldCounter UInt8, IsEvent UInt8, IsParameter UInt8, DontCountHits UInt8, WithHash UInt8, HitColor FixedString(1), UTCEventTime DateTime, Age UInt8, Sex UInt8, Income UInt8, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), RemoteIP UInt32, RemoteIP6 FixedString(16), WindowName Int32, OpenerName Int32, HistoryLength Int16, BrowserLanguage FixedString(2), BrowserCountry FixedString(2), SocialNetwork String, SocialAction String, HTTPError UInt16, SendTiming Int32, DNSTiming Int32, ConnectTiming Int32, ResponseStartTiming Int32, ResponseEndTiming Int32, FetchTiming Int32, RedirectTiming Int32, DOMInteractiveTiming Int32, DOMContentLoadedTiming Int32, DOMCompleteTiming Int32, LoadEventStartTiming Int32, LoadEventEndTiming Int32, NSToDOMContentLoadedTiming Int32, FirstPaintTiming Int32, RedirectCount Int8, SocialSourceNetworkID UInt8, SocialSourcePage String, ParamPrice Int64, ParamOrderID String, ParamCurrency FixedString(3), ParamCurrencyID UInt16, GoalsReached Array(UInt32), OpenstatServiceName String, OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String, UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, RefererHash UInt64, URLHash UInt64, CLID UInt32, YCLID UInt64, ShareService String, ShareURL String, ShareTitle String, ParsedParams Nested(Key1 String, Key2 String, Key3 String, Key4 String, Key5 String, ValueDouble Float64), IslandID FixedString(16), RequestNum UInt32, RequestTry UInt8) ENGINE = MergeTree()
PARTITION BY toYYYYMM(EventDate) ORDER BY (CounterID, EventDate, intHash32(UserID)) SAMPLE BY
intHash32(UserID) SETTINGS index_granularity = 8192"
## import data
cat hits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.hits_v1 FORMAT TSV" --max_insert_block_size=100000
## optionally you can optimize table
clickhouse-client --query "OPTIMIZE TABLE datasets.hits_v1 FINAL"
clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1"

```

从压缩tsv文件下载和导入访问：

```

curl https://clickhouse-datasets.s3.yandex.net/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
## now create table
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
clickhouse-client --query "CREATE TABLE datasets.visits_v1 ( CounterID UInt32, StartDate Date, Sign Int8, IsNew UInt8, VisitID UInt64, UserID UInt64, StartTime DateTime, Duration UInt32, UTCStartTime DateTime, PageViews Int32, Hits Int32, IsBounce UInt8, Referer String, StartURL String, RefererDomain String, StartURLDomain String, EndURL String, LinkURL String, IsDownload UInt8, TraficSourceID UInt8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8, PlaceID Int32, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions Array(UInt32), IsYandex UInt8, GoalReachesDepth Int32, GoalReachesURL Int32, GoalReachesAny Int32, SocialSourceNetworkID UInt8, SocialSourcePage String, MobilePhoneModel String, ClientEventTime DateTime, RegionID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RemoteIP UInt32, RemoteIP6 FixedString(16), IPNetworkID UInt32, SilverlightVersion3 UInt32, CodeVersion UInt32, ResolutionWidth UInt16, ResolutionHeight UInt16, UserAgentMajor UInt16, UserAgentMinor UInt16, WindowClientWidth UInt16, WindowClientHeight UInt16, SilverlightVersion2 UInt8, SilverlightVersion4 UInt16, FlashVersion3 UInt16, FlashVersion4 UInt16, ClientTimeZone Int16, OS UInt8, UserAgent UInt8, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, NetMajor UInt8, NetMinor UInt8, MobilePhone UInt8, SilverlightVersion1 UInt8, Age UInt8, Sex UInt8, Income UInt8, JavaEnable UInt8, CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, BrowserLanguage UInt16, BrowserCountry UInt16, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), Params Array(String), Goals Nested(ID UInt32, Serial UInt32, EventTime DateTime, Price Int64, OrderID String, CurrencyID UInt32), WatchIDs Array(UInt64), ParamSumPrice Int64, ParamCurrency FixedString(3), ParamCurrencyID UInt16, ClickLogID UInt64, ClickEventID Int32, ClickGoodEvent Int32, ClickEventTime DateTime, ClickPriorityID Int32, ClickPhraseID Int32, ClickPageID Int32, ClickPlaceID Int32, ClickTypeID Int32, ClickResourceID Int32, ClickCost UInt32, ClickClientIP UInt32, ClickDomainID UInt32, ClickURL String, ClickAttempt UInt8, ClickOrderID UInt32, ClickParamID UInt32

```

```

UInt32, ClickDomainID UInt32, ClickURL String, ClickAttempt UInt8, ClickOrderID UInt32, ClickBannerID UInt32,
ClickMarketCategoryID UInt32, ClickMarketPP UInt32, ClickMarketCategoryName String, ClickMarketPPName String,
ClickAWAPSCampaignName String, ClickPageName String, ClickTargetType UInt16, ClickTargetPhraseID UInt64,
ClickContextType UInt8, ClickSelectType Int8, ClickOptions String, ClickGroupBannerID Int32, OpenstatServiceName
String, OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium
String, UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, FirstVisit
DateTime, PredLastVisit Date, LastVisit Date, TotalVisits UInt32, TraficSource Nested(ID Int8, SearchEngineID
UInt16, AdvEngineID UInt8, PlaceID UInt16, SocialSourceNetworkID UInt8, Domain String, SearchPhrase String,
SocialSourcePage String), Attendance FixedString(16), CLID UInt32, YCLID UInt64, NormalizedRefererHash UInt64,
SearchPhraseHash UInt64, RefererDomainHash UInt64, NormalizedStartURLHash UInt64, StartURLDomainHash UInt64,
NormalizedEndURLHash UInt64, TopLevelDomain UInt64, URLScheme UInt64, OpenstatServiceNameHash UInt64,
OpenstatCampaignIDHash UInt64, OpenstatAdIDHash UInt64, OpenstatSourceIDHash UInt64, UTMSourceHash UInt64,
UTMMediumHash UInt64, UTMCampaignHash UInt64, UTMContentHash UInt64, UTMTermHash UInt64, FromHash
UInt64, WebVisorEnabled UInt8, WebVisorActivity UInt32, ParsedParams Nested(Key1 String, Key2 String, Key3
String, Key4 String, Key5 String, ValueDouble Float64), Market Nested(Type UInt8, GoalID UInt32, OrderID String,
OrderPrice Int64, PP UInt32, DirectPlaceID UInt32, DirectOrderID UInt32, DirectBannerID UInt32, GoodID String,
GoodName String, GoodQuantity Int32, GoodPrice Int64), IslandID FixedString(16)) ENGINE =
CollapsingMergeTree(Sign) PARTITION BY toYYYYMM(StartDate) ORDER BY (CounterID, StartDate, intHash32(UserID),
VisitID) SAMPLE BY intHash32(UserID) SETTINGS index_granularity = 8192"
## import data
cat visits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.visits_v1 FORMAT TSV" --
max_insert_block_size=100000
## optionally you can optimize table
clickhouse-client --query "OPTIMIZE TABLE datasets.visits_v1 FINAL"
clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1"

```

查询示例

[点击教程](#) 是基于Yandex的。Metrica数据集和开始使用此数据集的推荐方式是通过教程。

查询这些表的其他示例可以在 [有状态测试 ClickHouse的](#)（它们被命名为 `test.hists` 和 `test.visits` 那里）。

AMPLab大数据基准测试

参考 <https://amplab.cs.berkeley.edu/benchmark/>

需要您在<https://aws.amazon.com>注册一个免费的账号。注册时需要您提供信用卡、邮箱、电话等信息。之后可以在https://console.aws.amazon.com/iam/home?nc2=h_m_sc#security_credential获取新的访问密钥

在控制台运行以下命令：

```

$ sudo apt-get install s3cmd
$ mkdir tiny; cd tiny;
$ s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/tiny/ .
$ cd ..
$ mkdir 1node; cd 1node;
$ s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/1node/ .
$ cd ..
$ mkdir 5nodes; cd 5nodes;
$ s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/5nodes/ .
$ cd ..

```

在ClickHouse运行如下查询：

```

CREATE TABLE rankings_tiny
(
    pageURL String,
    pageRank UInt32,
    avgDuration UInt32
) ENGINE = Log

```

```

) ENGINE = Log;

CREATE TABLE uservisits_tiny
(
    sourceIP String,
    destinationURL String,
    visitDate Date,
    adRevenue Float32,
    UserAgent String,
    cCode FixedString(3),
    lCode FixedString(6),
    searchWord String,
    duration UInt32
) ENGINE = MergeTree(visitDate, visitDate, 8192);

CREATE TABLE rankings_1node
(
    pageURL String,
    pageRank UInt32,
    avgDuration UInt32
) ENGINE = Log;

CREATE TABLE uservisits_1node
(
    sourceIP String,
    destinationURL String,
    visitDate Date,
    adRevenue Float32,
    UserAgent String,
    cCode FixedString(3),
    lCode FixedString(6),
    searchWord String,
    duration UInt32
) ENGINE = MergeTree(visitDate, visitDate, 8192);

CREATE TABLE rankings_5nodes_on_single
(
    pageURL String,
    pageRank UInt32,
    avgDuration UInt32
) ENGINE = Log;

CREATE TABLE uservisits_5nodes_on_single
(
    sourceIP String,
    destinationURL String,
    visitDate Date,
    adRevenue Float32,
    UserAgent String,
    cCode FixedString(3),
    lCode FixedString(6),
    searchWord String,
    duration UInt32
) ENGINE = MergeTree(visitDate, visitDate, 8192);

```

回到控制台运行如下命令：

```

$ for i in tiny/rankings/*.deflate; do echo $i | zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO rankings_tiny FORMAT CSV"; done
$ for i in tiny/uservisits/*.deflate; do echo $i | zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO uservisits_tiny FORMAT CSV"; done
$ for i in 1node/rankings/*.deflate; do echo $i | zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --

```

```

$ for i in 1node/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO rankings_1node FORMAT CSV"; done
$ for i in 1node/uservisits/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO uservisits_1node FORMAT CSV"; done
$ for i in 5nodes/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO rankings_5nodes_on_single FORMAT CSV"; done
$ for i in 5nodes/uservisits/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perf01j --query="INSERT INTO uservisits_5nodes_on_single FORMAT CSV"; done

```

简单的查询示例：

```

SELECT pageURL, pageRank FROM rankings_1node WHERE pageRank > 1000

SELECT substring(sourceIP, 1, 8), sum(adRevenue) FROM uservisits_1node GROUP BY substring(sourceIP, 1, 8)

SELECT
    sourceIP,
    sum(adRevenue) AS totalRevenue,
    avg(pageRank) AS pageRank
FROM rankings_1node ALL INNER JOIN
(
    SELECT
        sourceIP,
        destinationURL AS pageURL,
        adRevenue
    FROM uservisits_1node
    WHERE (visitDate > '1980-01-01') AND (visitDate < '1980-04-01')
) USING pageURL
GROUP BY sourceIP
ORDER BY totalRevenue DESC
LIMIT 1

```

Criteo TB级别点击日志

可以从<http://labs.criteo.com/downloads/download-terabyte-click-logs/>上下载数据

创建原始数据对应的表结构：

```

CREATE TABLE criteo_log (date Date, clicked UInt8, int1 Int32, int2 Int32, int3 Int32, int4 Int32, int5 Int32, int6 Int32, int7
Int32, int8 Int32, int9 Int32, int10 Int32, int11 Int32, int12 Int32, int13 Int32, cat1 String, cat2 String, cat3 String, cat4
String, cat5 String, cat6 String, cat7 String, cat8 String, cat9 String, cat10 String, cat11 String, cat12 String, cat13
String, cat14 String, cat15 String, cat16 String, cat17 String, cat18 String, cat19 String, cat20 String, cat21 String, cat22
String, cat23 String, cat24 String, cat25 String, cat26 String) ENGINE = Log

```

下载数据：

```

$ for i in {00..23}; do echo $i; zcat datasets/criteo/day_${i#0}.gz | sed -r 's/^2000-01-$i/00/24'\t' | clickhouse-client --host=example-perf01j --query="INSERT INTO criteo_log FORMAT TabSeparated"; done

```

创建转换后的数据对应的表结构：

```

CREATE TABLE criteo
(
    date Date,
    clicked UInt8,
    int1 Int32,
    int2 Int32,

```

```

int3 Int32,
int4 Int32,
int5 Int32,
int6 Int32,
int7 Int32,
int8 Int32,
int9 Int32,
int10 Int32,
int11 Int32,
int12 Int32,
int13 Int32,
icat1 UInt32,
icat2 UInt32,
icat3 UInt32,
icat4 UInt32,
icat5 UInt32,
icat6 UInt32,
icat7 UInt32,
icat8 UInt32,
icat9 UInt32,
icat10 UInt32,
icat11 UInt32,
icat12 UInt32,
icat13 UInt32,
icat14 UInt32,
icat15 UInt32,
icat16 UInt32,
icat17 UInt32,
icat18 UInt32,
icat19 UInt32,
icat20 UInt32,
icat21 UInt32,
icat22 UInt32,
icat23 UInt32,
icat24 UInt32,
icat25 UInt32,
icat26 UInt32
) ENGINE = MergeTree(date, intHash32(icat1), (date, intHash32(icat1)), 8192)

```

将第一张表中的原始数据转化写入到第二张表中去：

```

INSERT INTO criteo SELECT date, clicked, int1, int2, int3, int4, int5, int6, int7, int8, int9, int10, int11, int12, int13,
reinterpretAsUInt32(unhex(cat1)) AS icat1, reinterpretAsUInt32(unhex(cat2)) AS icat2, reinterpretAsUInt32(unhex(cat3))
AS icat3, reinterpretAsUInt32(unhex(cat4)) AS icat4, reinterpretAsUInt32(unhex(cat5)) AS icat5,
reinterpretAsUInt32(unhex(cat6)) AS icat6, reinterpretAsUInt32(unhex(cat7)) AS icat7, reinterpretAsUInt32(unhex(cat8))
AS icat8, reinterpretAsUInt32(unhex(cat9)) AS icat9, reinterpretAsUInt32(unhex(cat10)) AS icat10,
reinterpretAsUInt32(unhex(cat11)) AS icat11, reinterpretAsUInt32(unhex(cat12)) AS icat12,
reinterpretAsUInt32(unhex(cat13)) AS icat13, reinterpretAsUInt32(unhex(cat14)) AS icat14,
reinterpretAsUInt32(unhex(cat15)) AS icat15, reinterpretAsUInt32(unhex(cat16)) AS icat16,
reinterpretAsUInt32(unhex(cat17)) AS icat17, reinterpretAsUInt32(unhex(cat18)) AS icat18,
reinterpretAsUInt32(unhex(cat19)) AS icat19, reinterpretAsUInt32(unhex(cat20)) AS icat20,
reinterpretAsUInt32(unhex(cat21)) AS icat21, reinterpretAsUInt32(unhex(cat22)) AS icat22,
reinterpretAsUInt32(unhex(cat23)) AS icat23, reinterpretAsUInt32(unhex(cat24)) AS icat24,
reinterpretAsUInt32(unhex(cat25)) AS icat25, reinterpretAsUInt32(unhex(cat26)) AS icat26 FROM criteo_log;

DROP TABLE criteo_log;

```

星型基准测试

编译 dbgen:

```
$ git clone git@github.com:vadimtk/ssb-dbggen.git  
$ cd ssb-dbggen  
$ make
```

开始生成数据：

```
$ ./dbgen -s 1000 -T c  
$ ./dbgen -s 1000 -T l  
$ ./dbgen -s 1000 -T p  
$ ./dbgen -s 1000 -T s  
$ ./dbgen -s 1000 -T d
```

在ClickHouse中创建表结构：

```
CREATE TABLE customer  
(  
    C_CUSTKEY      UInt32,  
    C_NAME         String,  
    C_ADDRESS       String,  
    C_CITY          LowCardinality(String),  
    C_NATION        LowCardinality(String),  
    C_REGION        LowCardinality(String),  
    C_PHONE         String,  
    C_MKTSEGMENT   LowCardinality(String)  
)  
ENGINE = MergeTree ORDER BY (C_CUSTKEY);  
  
CREATE TABLE lineorder  
(  
    LO_ORDERKEY      UInt32,  
    LO_LINENUMBER    UInt8,  
    LO_CUSTKEY       UInt32,  
    LO_PARTKEY       UInt32,  
    LO_SUPPKEY       UInt32,  
    LO_ORDERDATE     Date,  
    LO_ORDERPRIORITY LowCardinality(String),  
    LO_SHIPPRIORITY  UInt8,  
    LO_QUANTITY      UInt8,  
    LO_EXTENDEDPRICE UInt32,  
    LO_ORDTOTALPRICE UInt32,  
    LO_DISCOUNT      UInt8,  
    LO_REVENUE       UInt32,  
    LO_SUPPLYCOST    UInt32,  
    LO_TAX           UInt8,  
    LO_COMMITDATE    Date,  
    LO_SHIPMODE      LowCardinality(String)  
)  
ENGINE = MergeTree PARTITION BY toYear(LO_ORDERDATE) ORDER BY (LO_ORDERDATE, LO_ORDERKEY);  
  
CREATE TABLE part  
(  
    P_PARTKEY      UInt32,  
    P_NAME         String,  
    P_MFGR          LowCardinality(String),  
    P_CATEGORY      LowCardinality(String),  
    P_BRAND         LowCardinality(String),  
    P_COLOR         LowCardinality(String),  
    P_TYPE          LowCardinality(String),
```

```

P_SIZE      UInt8,
P_CONTAINER  LowCardinality(String)
)
ENGINE = MergeTree ORDER BY P_PARTKEY;

CREATE TABLE supplier
(
    S_SUPPKEY    UInt32,
    S_NAME       String,
    S_ADDRESS    String,
    S_CITY       LowCardinality(String),
    S_NATION     LowCardinality(String),
    S_REGION     LowCardinality(String),
    S_PHONE      String
)
ENGINE = MergeTree ORDER BY S_SUPPKEY;

```

写入数据：

```

$ clickhouse-client --query "INSERT INTO customer FORMAT CSV" < customer.tbl
$ clickhouse-client --query "INSERT INTO part FORMAT CSV" < part.tbl
$ clickhouse-client --query "INSERT INTO supplier FORMAT CSV" < supplier.tbl
$ clickhouse-client --query "INSERT INTO lineorder FORMAT CSV" < lineorder.tbl

```

将《星型模型》转换为非规范化的《平面模型》：

```

SET max_memory_usage = 20000000000, allow_experimental_multiple_joins_emulation = 1;

CREATE TABLE lineorder_flat
ENGINE = MergeTree
PARTITION BY toYear(LO_ORDERDATE)
ORDER BY (LO_ORDERDATE, LO_ORDERKEY) AS
SELECT l.*, c.*, s.*, p.*
FROM lineorder l
ANY INNER JOIN customer c ON (c.C_CUSTKEY = l.LO_CUSTKEY)
ANY INNER JOIN supplier s ON (s.S_SUPPKEY = l.LO_SUPPKEY)
ANY INNER JOIN part p ON (p.P_PARTKEY = l.LO_PARTKEY);

ALTER TABLE lineorder_flat DROP COLUMN C_CUSTKEY, DROP COLUMN S_SUPPKEY, DROP COLUMN P_PARTKEY;

```

运行查询：

Q1.1

```

SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue FROM lineorder_flat WHERE toYear(LO_ORDERDATE) =
1993 AND LO_DISCOUNT BETWEEN 1 AND 3 AND LO_QUANTITY < 25;

```

Q1.2

```

SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue FROM lineorder_flat WHERE toYYYYMM(LO_ORDERDATE) =
199401 AND LO_DISCOUNT BETWEEN 4 AND 6 AND LO_QUANTITY BETWEEN 26 AND 35;

```

Q1.3

```

SELECT sum(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue FROM lineorder_flat WHERE toISOWeek(LO_ORDERDATE) =
6 AND toYear(LO_ORDERDATE) = 1994 AND LO_DISCOUNT BETWEEN 5 AND 7 AND LO_QUANTITY BETWEEN 26 AND

```

35;

Q2.1

```
SELECT sum(LO_REVENUE), toYear(LO_ORDERDATE) AS year, P_BRAND FROM lineorder_flat WHERE P_CATEGORY = 'MFGR#12' AND S_REGION = 'AMERICA' GROUP BY year, P_BRAND ORDER BY year, P_BRAND;
```

Q2.2

```
SELECT sum(LO_REVENUE), toYear(LO_ORDERDATE) AS year, P_BRAND FROM lineorder_flat WHERE P_BRAND BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND S_REGION = 'ASIA' GROUP BY year, P_BRAND ORDER BY year, P_BRAND;
```

Q2.3

```
SELECT sum(LO_REVENUE), toYear(LO_ORDERDATE) AS year, P_BRAND FROM lineorder_flat WHERE P_BRAND = 'MFGR#2239' AND S_REGION = 'EUROPE' GROUP BY year, P_BRAND ORDER BY year, P_BRAND;
```

Q3.1

```
SELECT C_NATION, S_NATION, toYear(LO_ORDERDATE) AS year, sum(LO_REVENUE) AS revenue FROM lineorder_flat WHERE C_REGION = 'ASIA' AND S_REGION = 'ASIA' AND year >= 1992 AND year <= 1997 GROUP BY C_NATION, S_NATION, year ORDER BY year asc, revenue desc;
```

Q3.2

```
SELECT C_CITY, S_CITY, toYear(LO_ORDERDATE) AS year, sum(LO_REVENUE) AS revenue FROM lineorder_flat WHERE C_NATION = 'UNITED STATES' AND S_NATION = 'UNITED STATES' AND year >= 1992 AND year <= 1997 GROUP BY C_CITY, S_CITY, year ORDER BY year asc, revenue desc;
```

Q3.3

```
SELECT C_CITY, S_CITY, toYear(LO_ORDERDATE) AS year, sum(LO_REVENUE) AS revenue FROM lineorder_flat WHERE (C_CITY = 'UNITED KI1' OR C_CITY = 'UNITED KI5') AND (S_CITY = 'UNITED KI1' OR S_CITY = 'UNITED KI5') AND year >= 1992 AND year <= 1997 GROUP BY C_CITY, S_CITY, year ORDER BY year asc, revenue desc;
```

Q3.4

```
SELECT C_CITY, S_CITY, toYear(LO_ORDERDATE) AS year, sum(LO_REVENUE) AS revenue FROM lineorder_flat WHERE (C_CITY = 'UNITED KI1' OR C_CITY = 'UNITED KI5') AND (S_CITY = 'UNITED KI1' OR S_CITY = 'UNITED KI5') AND toYYYYMM(LO_ORDERDATE) = '199712' GROUP BY C_CITY, S_CITY, year ORDER BY year asc, revenue desc;
```

Q4.1

```
SELECT toYear(LO_ORDERDATE) AS year, C_NATION, sum(LO_REVENUE - LO_SUPPLYCOST) AS profit FROM lineorder_flat WHERE C_REGION = 'AMERICA' AND S_REGION = 'AMERICA' AND (P_MFGR = 'MFGR#1' OR P_MFGR = 'MFGR#2') GROUP BY year, C_NATION ORDER BY year, C_NATION;
```

Q4.2

```
SELECT toYear(LO_ORDERDATE) AS year, S_NATION, P_CATEGORY, sum(LO_REVENUE - LO_SUPPLYCOST) AS profit FROM lineorder_flat WHERE C_REGION = 'AMERICA' AND S_REGION = 'AMERICA' AND (year = 1997 OR year = 1998) AND
```

```
(P_MFGR = 'MFGR#1' OR P_MFGR = 'MFGR#2') GROUP BY year, S_NATION, P_CATEGORY ORDER BY year, S_NATION, P_CATEGORY;
```

Q4.3

```
SELECT toYear(LO_ORDERDATE) AS year, S_CITY, P_BRAND, sum(LO_REVENUE - LO_SUPPLYCOST) AS profit FROM lineorder_flat WHERE S_NATION = 'UNITED STATES' AND (year = 1997 OR year = 1998) AND P_CATEGORY = 'MFGR#14' GROUP BY year, S_CITY, P_BRAND ORDER BY year, S_CITY, P_BRAND;
```

纽约市出租车数据

纽约市出租车数据有以下两个方式获取：

从原始数据导入

下载预处理好的分区数据

怎样导入原始数据

可以参考<https://github.com/toddwschneider/nyc-taxi-data>和<http://tech.marksblogg.com/billion-nyc-taxi-rides-redshift.html>中的关于数据集结构描述与数据下载指令说明。

数据集包含227GB的CSV文件。这大约需要一个小时的下载时间(1Gbit带宽下，并行下载大概是一半时间)。

下载时注意损坏的文件。可以检查文件大小并重新下载损坏的文件。

有些文件中包含一些无效的行，您可以使用如下语句修复他们：

```
sed -E '/(.*){18,}/d' data/yellow_tripdata_2010-02.csv > data/yellow_tripdata_2010-02.csv_
sed -E '/(.*){18,}/d' data/yellow_tripdata_2010-03.csv > data/yellow_tripdata_2010-03.csv_
mv data/yellow_tripdata_2010-02.csv_ data/yellow_tripdata_2010-02.csv
mv data/yellow_tripdata_2010-03.csv_ data/yellow_tripdata_2010-03.csv
```

然后您必须在PostgreSQL中预处理这些数据。这将创建多边形中的点（以匹配在地图中纽约市中范围），然后通过使用JOIN查询将数据关联组合到一个规范的表中。为了完成这部分操作，您需要安装PostgreSQL的同时安装PostGIS插件。

运行initialize_database.sh时要小心，并手动重新检查是否正确创建了所有表。

在PostgreSQL中处理每个月的数据大约需要20-30分钟，总共大约需要48小时。

您可以按如下方式检查下载的行数：

```
$ time psql nyc-taxi-data -c "SELECT count(*) FROM trips;"#
## Count
1298979494
(1 row)

real    7m9.164s
```

(根据Mark Litwintschik的系列博客报道数据略多余11亿行)

PostgreSQL处理这些数据大概需要370GB的磁盘空间。

从PostgreSQL中导出数据：

```
COPY
(
  SELECT trips.id,
  trips.vendor_id,
```

```

trips.pickup_datetime,
trips.dropoff_datetime,
trips.store_and_fwd_flag,
trips.rate_code_id,
trips.pickup_longitude,
trips.pickup_latitude,
trips.dropoff_longitude,
trips.dropoff_latitude,
trips.passenger_count,
trips.trip_distance,
trips.fare_amount,
trips.extra,
trips.mta_tax,
trips.tip_amount,
trips.tolls_amount,
trips.ehail_fee,
trips.improvement_surcharge,
trips.total_amount,
trips.payment_type,
trips.trip_type,
trips.pickup,
trips.dropoff,

cab_types.type cab_type,

weather.precipitation_tenths_of_mm rain,
weather.snow_depth_mm,
weather.snowfall_mm,
weather.max_temperature_tenths_degrees_celsius max_temp,
weather.min_temperature_tenths_degrees_celsius min_temp,
weather.average_wind_speed_tenths_of_meters_per_second wind,

pick_up.gid pickup_nyct2010_gid,
pick_up.ctlabel pickup_ctlabel,
pick_up.borocode pickup_borocode,
pick_up.boroname pickup_boroname,
pick_up.ct2010 pickup_ct2010,
pick_up.boroc2010 pickup_boroc2010,
pick_up.cdeligibil pickup_cdeligibil,
pick_up.ntacode pickup_ntacode,
pick_up.ntaname pickup_ntaname,
pick_up.puma pickup_puma,

drop_off.gid dropoff_nyct2010_gid,
drop_off.ctlabel dropoff_ctlabel,
drop_off.borocode dropoff_borocode,
drop_off.boroname dropoff_boroname,
drop_off.ct2010 dropoff_ct2010,
drop_off.boroc2010 dropoff_boroc2010,
drop_off.cdeligibil dropoff_cdeligibil,
drop_off.ntacode dropoff_ntacode,
drop_off.ntaname dropoff_ntaname,
drop_off.puma dropoff_puma

FROM trips
LEFT JOIN cab_types
    ON trips.cab_type_id = cab_types.id
LEFT JOIN central_park_weather_observations_raw weather
    ON weather.date = trips.pickup_datetime::date
LEFT JOIN nyct2010 pick_up
    ON pick_up.gid = trips.pickup_nyct2010_gid
LEFT JOIN nyct2010 drop_off
    ON drop_off.gid = trips.dropoff_nyct2010_gid

```

```
) TO '/opt/milovidov/nyc-taxi-data/trips.tsv';
```

数据快照的创建速度约为每秒50 MB。在创建快照时，PostgreSQL以每秒约28 MB的速度从磁盘读取数据。
这大约需要5个小时。最终生成的TSV文件为590612904969 bytes。

在ClickHouse中创建临时表：

```
CREATE TABLE trips
(
    trip_id          UInt32,
    vendor_id        String,
    pickup_datetime  DateTime,
    dropoff_datetime Nullable(DateTime),
    store_and_fwd_flag Nullable(FixedString(1)),
    rate_code_id     Nullable(UInt8),
    pickup_longitude Nullable(Float64),
    pickup_latitude  Nullable(Float64),
    dropoff_longitude Nullable(Float64),
    dropoff_latitude Nullable(Float64),
    passenger_count Nullable(UInt8),
    trip_distance    Nullable(Float64),
    fare_amount      Nullable(Float32),
    extra            Nullable(Float32),
    mta_tax          Nullable(Float32),
    tip_amount       Nullable(Float32),
    tolls_amount     Nullable(Float32),
    ehail_fee        Nullable(Float32),
    improvement_surcharge Nullable(Float32),
    total_amount     Nullable(Float32),
    payment_type     Nullable(String),
    trip_type        Nullable(UInt8),
    pickup           Nullable(String),
    dropoff          Nullable(String),
    cab_type         Nullable(String),
    precipitation    Nullable(UInt8),
    snow_depth       Nullable(UInt8),
    snowfall         Nullable(UInt8),
    max_temperature  Nullable(UInt8),
    min_temperature  Nullable(UInt8),
    average_wind_speed Nullable(UInt8),
    pickup_nyct2010_gid Nullable(UInt8),
    pickup_ctlabel   Nullable(String),
    pickup_borocode  Nullable(UInt8),
    pickup_boroname  Nullable(String),
    pickup_ct2010    Nullable(String),
    pickup_boroct2010 Nullable(String),
    pickup_cdeligibil Nullable(FixedString(1)),
    pickup_ntacode   Nullable(String),
    pickup_ntaname   Nullable(String),
    pickup_puma      Nullable(String),
    dropoff_nyct2010_gid Nullable(UInt8),
    dropoff_ctlabel  Nullable(String),
    dropoff_borocode Nullable(UInt8),
    dropoff_boroname Nullable(String),
    dropoff_ct2010    Nullable(String),
    dropoff_boroct2010 Nullable(String),
    dropoff_cdeligibil Nullable(String),
    dropoff_ntacode  Nullable(String),
    dropoff_ntaname  Nullable(String),
    dropoff_puma     Nullable(String)
) ENGINE = Log;
```

接下来,需要将字段转换为更正确的数据类型,并且在可能的情况下,消除NULL。

```
$ time clickhouse-client --query="INSERT INTO trips FORMAT TabSeparated" < trips.tsv
real 75m56.214s
```

数据的读取速度为112-140 Mb/秒。

通过这种方式将数据加载到Log表中需要76分钟。

这个表中的数据需要使用142 GB的磁盘空间.

(也可以直接使用COPY ... TO PROGRAM从Postgres中导入数据)

由于数据中与天气相关的所有数据(precipitation.....average_wind_speed)都填充了NULL。所以,我们将从最终数据集中删除它们

首先,我们使用单台服务器创建表,后面我们将在多台节点上创建这些表。

创建表结构并写入数据:

```
CREATE TABLE trips_mergetree
ENGINE = MergeTree(pickup_date, pickup_datetime, 8192)
AS SELECT
    trip_id,
    CAST(vendor_id AS Enum8('1' = 1, '2' = 2, 'CMT' = 3, 'VTS' = 4, 'DDS' = 5, 'B02512' = 10, 'B02598' = 11, 'B02617' = 12,
    'B02682' = 13, 'B02764' = 14)) AS vendor_id,
    toDate(pickup_datetime) AS pickup_date,
    ifNull(pickup_datetime, toDateTime(0)) AS pickup_datetime,
    toDate(dropoff_datetime) AS dropoff_date,
    ifNull(dropoff_datetime, toDateTime(0)) AS dropoff_datetime,
    assumeNotNull(store_and_fwd_flag) IN ('Y', '1', '2') AS store_and_fwd_flag,
    assumeNotNull(rate_code_id) AS rate_code_id,
    assumeNotNull(pickup_longitude) AS pickup_longitude,
    assumeNotNull(pickup_latitude) AS pickup_latitude,
    assumeNotNull(dropoff_longitude) AS dropoff_longitude,
    assumeNotNull(dropoff_latitude) AS dropoff_latitude,
    assumeNotNull(passenger_count) AS passenger_count,
    assumeNotNull(trip_distance) AS trip_distance,
    assumeNotNull(fare_amount) AS fare_amount,
    assumeNotNull(extra) AS extra,
    assumeNotNull(mta_tax) AS mta_tax,
    assumeNotNull(tip_amount) AS tip_amount,
    assumeNotNull(tolls_amount) AS tolls_amount,
    assumeNotNull(ehail_fee) AS ehail_fee,
    assumeNotNull(improvement_surcharge) AS improvement_surcharge,
    assumeNotNull(total_amount) AS total_amount,
    CAST((assumeNotNull(payment_type) AS pt) IN ('CSH', 'CASH', 'Cash', 'CAS', 'Cas', '1') ? 'CSH' : (pt IN ('CRD', 'Credit', 'Cre',
    'CRE', 'CREDIT', '2') ? 'CRE' : (pt IN ('NOC', 'No Charge', 'No', '3') ? 'NOC' : (pt IN ('DIS', 'Dispute', 'Dis', '4') ? 'DIS' : 'UNK'))))
    AS Enum8('CSH' = 1, 'CRE' = 2, 'UNK' = 0, 'NOC' = 3, 'DIS' = 4)) AS payment_type_,
    assumeNotNull(trip_type) AS trip_type,
    ifNull(toFixedString(unhex(pickup), 25), toFixedString('', 25)) AS pickup,
    ifNull(toFixedString(unhex(dropoff), 25), toFixedString('', 25)) AS dropoff,
    CAST(assumeNotNull(cab_type) AS Enum8('yellow' = 1, 'green' = 2, 'uber' = 3)) AS cab_type,
    assumeNotNull(pickup_nyct2010_gid) AS pickup_nyct2010_gid,
   toFloat32(ifNull(pickup_ctlabel, '0')) AS pickup_ctlabel,
    assumeNotNull(pickup_borocode) AS pickup_borocode,
    CAST(assumeNotNull(pickup_boroname) AS Enum8('Manhattan' = 1, 'Queens' = 4, 'Brooklyn' = 3, '' = 0, 'Bronx' = 2,
    'Staten Island' = 5)) AS pickup_boroname,
```

```

toFixedString(ifNull(pickup_ct2010, '000000'), 6) AS pickup_ct2010,
toFixedString(ifNull(pickup_boroct2010, '0000000'), 7) AS pickup_boroct2010,
CAST(assumeNotNull(ifNull(pickup_cdeligibil, ' ')) AS Enum8(' = 0, 'E' = 1, 'I' = 2)) AS pickup_cdeligibil,
toFixedString(ifNull(pickup_ntacode, '0000'), 4) AS pickup_ntacode,

CAST(assumeNotNull(pickup_ntaname) AS Enum16(" = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince)'s Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Clarendon-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotone Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Doulaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195) AS pickup_ntaname,

toUInt16(ifNull(pickup_puma, '0')) AS pickup_puma,

assumeNotNull(dropoff_nyct2010_gid) AS dropoff_nyct2010_gid,
toFloat32(ifNull(dropoff_ctlabell, '0')) AS dropoff_ctlabell,
assumeNotNull(dropoff_borocode) AS dropoff_borocode,
CAST(assumeNotNull(dropoff_boroname) AS Enum8('Manhattan' = 1, 'Queens' = 4, 'Brooklyn' = 3, " = 0, 'Bronx' = 2,

```

```

'Staten Island' = 5)) AS dropoff_boriname,
toFixedString(ifNull(dropoff_ct2010, '000000'), 6) AS dropoff_ct2010,
toFixedString(ifNull(dropoff_boroct2010, '0000000'), 7) AS dropoff_boroct2010,
CAST(assumeNotNull(ifNull(dropoff_cdeligibil, '')) AS Enum8(' ' = 0, 'E' = 1, 'I' = 2)) AS dropoff_cdeligibil,
toFixedString(ifNull(dropoff_ntacode, '0000'), 4) AS dropoff_ntacode,

CAST(assumeNotNull(dropoff_ntaname) AS Enum16(' ' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaslaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195)) AS dropoff_ntaname,
toUInt16(ifNull(dropoff_puma, '0')) AS dropoff_puma

```

FROM trips

这需要3030秒，速度约为每秒428,000行。

要加快速度，可以使用 Log 引擎替换'MergeTree`引擎来创建表。在这种情况下，下载速度超过200秒。

这个表需要使用126GB的磁盘空间。

```
SELECT formatReadableSize(sum(bytes)) FROM system.parts WHERE table = 'trips_mergetree' AND active
```

```
└─formatReadableSize(sum(bytes))─  
| 126.18 GiB |
```

除此之外，你还可以在MergeTree上运行OPTIMIZE查询来进行优化。但这不是必须的，因为即使在没有进行优化的情况下它的表现依然是很好的。

下载预处理好的分区数据

```
$ curl -O https://clickhouse-datasets.s3.yandex.net/trips_mergetree/partitions/trips_mergetree.tar  
$ tar xvf trips_mergetree.tar -C /var/lib/clickhouse # path to ClickHouse data directory  
$ # check permissions of unpacked data, fix if required  
$ sudo service clickhouse-server restart  
$ clickhouse-client --query "select count(*) from datasets.trips_mergetree"
```

信息

如果要运行下面的SQL查询，必须使用完整的表名，

datasets.trips_mergetree。

单台服务器运行结果

Q1:

```
SELECT cab_type, count(*) FROM trips_mergetree GROUP BY cab_type
```

0.490秒

Q2:

```
SELECT passenger_count, avg(total_amount) FROM trips_mergetree GROUP BY passenger_count
```

1.224秒

Q3:

```
SELECT passenger_count, toYear(pickup_date) AS year, count(*) FROM trips_mergetree GROUP BY passenger_count, year
```

2.104秒

Q4:

```
SELECT passenger_count, toYear(pickup_date) AS year, round(trip_distance) AS distance, count(*)  
FROM trips_mergetree
```

```
FROM trips_mergereee
GROUP BY passenger_count, year, distance
ORDER BY year, count(*) DESC
```

3.593秒

我们使用的是如下配置的服务器：

两个英特尔（R）至强（R）CPU E5-2650v2@2.60GHz，总共有16个物理内核，128GiB RAM，硬件RAID-5上的8X6TB HD

执行时间是取三次运行中最好的值，但是从第二次查询开始，查询就讲从文件系统的缓存中读取数据。同时在每次读取和处理后不在进行缓存。

在三台服务器中创建表结构：

在每台服务器中运行：

```
CREATE TABLE default.trips_mergetree_third ( trip_id UInt32, vendor_id Enum8('1' = 1, '2' = 2, 'CMT' = 3, 'VTS' = 4,
'DDS' = 5, 'B02512' = 10, 'B02598' = 11, 'B02617' = 12, 'B02682' = 13, 'B02764' = 14), pickup_date Date,
pickup_datetime DateTime, dropoff_date Date, dropoff_datetime DateTime, store_and_fwd_flag UInt8, rate_code_id
UInt8, pickup_longitude Float64, pickup_latitude Float64, dropoff_longitude Float64, dropoff_latitude Float64,
passenger_count UInt8, trip_distance Float64, fare_amount Float32, extra Float32, mta_tax Float32, tip_amount Float32,
tolls_amount Float32, ehail_fee Float32, improvement_surcharge Float32, total_amount Float32, payment_type_
Enum8('UNK' = 0, 'CSH' = 1, 'CRE' = 2, 'NOC' = 3, 'DIS' = 4), trip_type UInt8, pickup FixedString(25), dropoff
FixedString(25), cab_type Enum8('yellow' = 1, 'green' = 2, 'uber' = 3), pickup_nyct2010_gid UInt8, pickup_ctlabel
Float32, pickup_borocode UInt8, pickup_boroname Enum8('' = 0, 'Manhattan' = 1, 'Bronx' = 2, 'Brooklyn' = 3, 'Queens' =
4, 'Staten Island' = 5), pickup_ct2010 FixedString(6), pickup_boroct2010 FixedString(7), pickup_cdeligibil Enum8('' = 0,
'E' = 1, 'I' = 2), pickup_ntacode FixedString(4), pickup_ntaname Enum16('' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' =
2, 'Annadale-Huguenot-Prince\'s Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley
Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11,
'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16,
'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19,
'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23,
'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll
Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31,
'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Clarendon-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill'
= 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41,
'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44,
'Douglas Manor-Douglas-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East
Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52,
'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East
Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' =
61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest
Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70,
'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73,
'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78,
'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82,
'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West
Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90,
'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton'
= 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99,
'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-
Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' =
107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott
Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver
Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North
Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123,
'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-
Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-
Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' =
135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' =
```

133, 'Prospect Heights' = 133, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195), pickup_puma UInt16, dropoff_nyct2010_gid UInt8, dropoff_ctlabel Float32, dropoff_borocode UInt8, dropoff_boroname Enum8(" = 0, 'Manhattan' = 1, 'Bronx' = 2, 'Brooklyn' = 3, 'Queens' = 4, 'Staten Island' = 5), dropoff_ct2010 FixedString(6), dropoff_boroc2010 FixedString(7), dropoff_cdeligibil Enum8(" = 0, 'E' = 1, 'I' = 2), dropoff_ntacode FixedString(4), dropoff_ntaname Enum16(" = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince\''s Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Clarendon-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaslaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner\''s Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' =

```
174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195), dropoff_puma UInt16) ENGINE = MergeTree(pickup_date, pickup_datetime, 8192)
```

在之前的服务器中运行：

```
CREATE TABLE trips_mergetree_x3 AS trips_mergetree_third ENGINE = Distributed(perftest, default, trips_mergetree_third, rand())
```

运行如下查询重新分布数据：

```
INSERT INTO trips_mergetree_x3 SELECT * FROM trips_mergetree
```

这个查询需要运行2454秒。

在三台服务器集群中运行的结果：

Q1:0.212秒。

Q2 : 0.438秒。

Q3 : 0.733秒。

Q4:1.241秒。

不出意料，查询是线性扩展的。

我们同时在140台服务器的集群中运行的结果：

Q1 : 0.028秒。

Q2 : 0.043秒。

Q3 : 0.051秒。

Q4 : 0.072秒。

在这种情况下，查询处理时间首先由网络延迟确定。

我们使用位于芬兰的Yandex数据中心中的客户端去位于俄罗斯的集群上运行查询，这增加了大约20毫秒的延迟。

总结

服务器	Q1	Q2	Q3	Q4
1	0.490	1.224	2.104	3.593
3	0.212	0.438	0.733	1.241
140	0.028	0.043	0.051	0.072

维基访问数据

参考: <http://dumps.wikimedia.org/other/pagecounts-raw/>

创建表结构：

```
CREATE TABLE wikistat
(
    date Date,
```

```
time DateTime,  
project String,  
subproject String,  
path String,  
hits UInt64,  
size UInt64  
) ENGINE = MergeTree(date, (path, time), 8192);
```

加载数据：

```
$ for i in {2007..2016}; do for j in {01..12}; do echo $i-$j >&2; curl -sSL "http://dumps.wikimedia.org/other/pagecounts-raw/$i/$i-$j/" | grep -oE 'pagecounts-[0-9]+-[0-9]+\.\gz'; done; done | sort | uniq | tee links.txt
$ cat links.txt | while read link; do wget http://dumps.wikimedia.org/other/pagecounts-raw/$(echo $link | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})[0-9]{2}-[0-9]+\.\gz\|1\')/$(echo $link | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})[0-9]{2}-[0-9]+\.\gz\|1-\2\')/$link; done
$ ls -1 /opt/wikistat/ | grep gz | while read i; do echo $i; gzip -cd /opt/wikistat/$i | ./wikistat-loader --time="$(echo -n $i | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})([0-9]{2})-([0-9]{2})([0-9]{2})([0-9]{2}).gz\|1-\2-\3 \4-00-00/')" | clickhouse-client --query="INSERT INTO wikistat FORMAT TabSeparated"; done
```

航班飞行数据

航班飞行数据有以下两个方式获取：

- 从原始数据导入
 - 下载预处理好的分区数据

从原始数据导入

下载数据：

```
for s in `seq 1987 2018`  
do  
for m in `seq 1 12`  
do  
wget https://transtats.bts.gov/PREZIP/On_Time_Reportng_Carrier_On_Time_Performance_1987_present_${s}_${m}.zip  
done  
done
```

(引用 <https://github.com/Percona-Lab/ontime-airline-performance/blob/master/download.sh>)

创建表结构：

```
CREATE TABLE `ontime` (  
    `Year` UInt16,  
    `Quarter` UInt8,  
    `Month` UInt8,  
    `DayofMonth` UInt8,  
    `DayOfWeek` UInt8,  
    `FlightDate` Date,  
    `UniqueCarrier` FixedString(7),  
    `AirlineID` Int32,  
    `Carrier` FixedString(2),  
    `TailNum` String,  
    `FlightNum` String,  
    `OriginAirportID` Int32,  
    `OriginAirportSeqID` Int32,  
    `OriginCityMarketID` Int32,  
    `Origin` FixedString(5),
```

```
`OriginCityName` String,  
`OriginState` FixedString(2),  
`OriginStateFips` String,  
`OriginStateName` String,  
`OriginWac` Int32,  
`DestAirportID` Int32,  
`DestAirportSeqID` Int32,  
`DestCityMarketID` Int32,  
`Dest` FixedString(5),  
`DestCityName` String,  
`DestState` FixedString(2),  
`DestStateFips` String,  
`DestStateName` String,  
`DestWac` Int32,  
`CRSDepTime` Int32,  
`DepTime` Int32,  
`DepDelay` Int32,  
`DepDelayMinutes` Int32,  
`DepDel15` Int32,  
`DepartureDelayGroups` String,  
`DepTimeBlk` String,  
`TaxiOut` Int32,  
`WheelsOff` Int32,  
`WheelsOn` Int32,  
`TaxiIn` Int32,  
`CRSArrTime` Int32,  
`ArrTime` Int32,  
`ArrDelay` Int32,  
`ArrDelayMinutes` Int32,  
`ArrDel15` Int32,  
`ArrivalDelayGroups` Int32,  
`ArrTimeBlk` String,  
`Cancelled` UInt8,  
`CancellationCode` FixedString(1),  
`Diverted` UInt8,  
`CRSElapsedTime` Int32,  
`ActualElapsedTime` Int32,  
`AirTime` Int32,  
`Flights` Int32,  
`Distance` Int32,  
`DistanceGroup` UInt8,  
`CarrierDelay` Int32,  
`WeatherDelay` Int32,  
`NASDelay` Int32,  
`SecurityDelay` Int32,  
`LateAircraftDelay` Int32,  
`FirstDepTime` String,  
`TotalAddGTime` String,  
`LongestAddGTime` String,  
`DivAirportLandings` String,  
`DivReachedDest` String,  
`DivActualElapsedTime` String,  
`DivArrDelay` String,  
`DivDistance` String,  
`Div1Airport` String,  
`Div1AirportID` Int32,  
`Div1AirportSeqID` Int32,  
`Div1WheelsOn` String,  
`Div1TotalGTime` String,  
`Div1LongestGTime` String,  
`Div1WheelsOff` String,  
`Div1TailNum` String,  
`Div2Airport` String
```

```
    `Div2Airport` String,
    `Div2AirportID` Int32,
    `Div2AirportSeqID` Int32,
    `Div2WheelsOn` String,
    `Div2TotalGTime` String,
    `Div2LongestGTime` String,
    `Div2WheelsOff` String,
    `Div2TailNum` String,
    `Div3Airport` String,
    `Div3AirportID` Int32,
    `Div3AirportSeqID` Int32,
    `Div3WheelsOn` String,
    `Div3TotalGTime` String,
    `Div3LongestGTime` String,
    `Div3WheelsOff` String,
    `Div3TailNum` String,
    `Div4Airport` String,
    `Div4AirportID` Int32,
    `Div4AirportSeqID` Int32,
    `Div4WheelsOn` String,
    `Div4TotalGTime` String,
    `Div4LongestGTime` String,
    `Div4WheelsOff` String,
    `Div4TailNum` String,
    `Div5Airport` String,
    `Div5AirportID` Int32,
    `Div5AirportSeqID` Int32,
    `Div5WheelsOn` String,
    `Div5TotalGTime` String,
    `Div5LongestGTime` String,
    `Div5WheelsOff` String,
    `Div5TailNum` String
) ENGINE = MergeTree
PARTITION BY Year
ORDER BY (Carrier, FlightDate)
SETTINGS index_granularity = 8192;
```

加载数据：

```
$ for i in *.zip; do echo $i; unzip -cq $i '*.csv' | sed 's/\.\.00//g' | clickhouse-client --host=example-perftest01j --
query="INSERT INTO ontimedata FORMAT CSVWithNames"; done
```

下载预处理好的分区数据

```
$ curl -O https://clickhouse-datasets.s3.yandex.net/ontimedata/partitions/ontimedata.tar
$ tar xvf ontimedata.tar -C /var/lib/clickhouse # path to ClickHouse data directory
$ # check permissions of unpacked data, fix if required
$ sudo service clickhouse-server restart
$ clickhouse-client --query "select count(*) from datasets.ontimedata"
```

信息

如果要运行下面的SQL查询，必须使用完整的表名，

datasets.ontimedata。

查询：

Q0.

```
SELECT avg(c1)
FROM
(
    SELECT Year, Month, count(*) AS c1
    FROM ontime
    GROUP BY Year, Month
);
```

Q1. 查询从2000年到2008年每天的航班数

```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE Year>=2000 AND Year<=2008
GROUP BY DayOfWeek
ORDER BY c DESC;
```

Q2. 查询从2000年到2008年每周延误超过10分钟的航班数。

```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE DepDelay>10 AND Year>=2000 AND Year<=2008
GROUP BY DayOfWeek
ORDER BY c DESC;
```

Q3. 查询2000年到2008年每个机场延误超过10分钟以上的次数

```
SELECT Origin, count(*) AS c
FROM ontime
WHERE DepDelay>10 AND Year>=2000 AND Year<=2008
GROUP BY Origin
ORDER BY c DESC
LIMIT 10;
```

Q4. 查询2007年各航空公司延误超过10分钟以上的次数

```
SELECT Carrier, count(*)
FROM ontime
WHERE DepDelay>10 AND Year=2007
GROUP BY Carrier
ORDER BY count(*) DESC;
```

Q5. 查询2007年各航空公司延误超过10分钟以上的百分比

```
SELECT Carrier, c, c2, c*100/c2 as c3
FROM
(
    SELECT
        Carrier,
        count(*) AS c
    FROM ontime
    WHERE DepDelay>10
        AND Year=2007
    GROUP BY Carrier
);
```

```
)  
JOIN  
(  
    SELECT  
        Carrier,  
        count(*) AS c2  
    FROM ontime  
    WHERE Year=2007  
    GROUP BY Carrier  
) USING Carrier  
ORDER BY c3 DESC;
```

更好的查询版本：

```
SELECT Carrier, avg(DepDelay>10)*100 AS c3  
FROM ontime  
WHERE Year=2007  
GROUP BY Carrier  
ORDER BY c3 DESC
```

Q6. 同上一个查询一致,只是查询范围扩大到2000年到2008年

```
SELECT Carrier, c, c2, c*100/c2 as c3  
FROM  
(  
    SELECT  
        Carrier,  
        count(*) AS c  
    FROM ontime  
    WHERE DepDelay>10  
        AND Year>=2000 AND Year<=2008  
    GROUP BY Carrier  
)  
JOIN  
(  
    SELECT  
        Carrier,  
        count(*) AS c2  
    FROM ontime  
    WHERE Year>=2000 AND Year<=2008  
    GROUP BY Carrier  
) USING Carrier  
ORDER BY c3 DESC;
```

更好的查询版本：

```
SELECT Carrier, avg(DepDelay>10)*100 AS c3  
FROM ontime  
WHERE Year>=2000 AND Year<=2008  
GROUP BY Carrier  
ORDER BY c3 DESC;
```

Q7. 每年航班延误超过10分钟的百分比

```
SELECT Year, c1/c2  
FROM  
(  
    SELECT
```

```
select
    Year,
    count(*)*100 as c1
from ontime
WHERE DepDelay>10
GROUP BY Year
)
JOIN
(
    select
        Year,
        count(*) as c2
    from ontime
    GROUP BY Year
) USING (Year)
ORDER BY Year;
```

更好的查询版本：

```
SELECT Year, avg(DepDelay>10)*100
FROM ontime
GROUP BY Year
ORDER BY Year;
```

Q8. 每年更受人们喜爱的目的地

```
SELECT DestCityName, uniqExact(OriginCityName) AS u
FROM ontime
WHERE Year >= 2000 and Year <= 2010
GROUP BY DestCityName
ORDER BY u DESC LIMIT 10;
```

Q9.

```
SELECT Year, count(*) AS c1
FROM ontime
GROUP BY Year;
```

Q10.

```
SELECT
    min(Year), max(Year), Carrier, count(*) AS cnt,
    sum(ArrDelayMinutes>30) AS flights_delayed,
    round(sum(ArrDelayMinutes>30)/count(*),2) AS rate
FROM ontime
WHERE
    DayOfWeek NOT IN (6,7) AND OriginState NOT IN ('AK', 'HI', 'PR', 'VI')
    AND DestState NOT IN ('AK', 'HI', 'PR', 'VI')
    AND FlightDate < '2010-01-01'
GROUP by Carrier
HAVING cnt>100000 and max(Year)>1990
ORDER by rate DESC
LIMIT 1000;
```

奖金：

```

SELECT avg(cnt)
FROM
(
  SELECT Year,Month,count(*) AS cnt
  FROM ontime
  WHERE DepDel15=1
  GROUP BY Year,Month
);
SELECT avg(c1) FROM
(
  SELECT Year,Month,count(*) AS c1
  FROM ontime
  GROUP BY Year,Month
);
SELECT DestCityName, uniqExact(OriginCityName) AS u
FROM ontime
GROUP BY DestCityName
ORDER BY u DESC
LIMIT 10;
SELECT OriginCityName, DestCityName, count() AS c
FROM ontime
GROUP BY OriginCityName, DestCityName
ORDER BY c DESC
LIMIT 10;
SELECT OriginCityName, count() AS c
FROM ontime
GROUP BY OriginCityName
ORDER BY c DESC
LIMIT 10;

```

这个性能测试由Vadim Tkachenko提供。参考：

- <https://www.percona.com/blog/2009/10/02/analyzing-air-traffic-performance-with-infobright-and-monetdb/>
- <https://www.percona.com/blog/2009/10/26/air-traffic-queries-in-luciddb/>
- <https://www.percona.com/blog/2009/11/02/air-traffic-queries-in-infinidb-early-alpha/>
- <https://www.percona.com/blog/2014/04/21/using-apache-hadoop-and-impala-together-with-mysql-for-data-analysis/>
- <https://www.percona.com/blog/2016/01/07/apache-spark-with-air-ontime-performance-data/>
- <http://nickmakos.blogspot.ru/2012/08/analyzing-air-traffic-performance-with.html>

ClickHouse体验平台

[ClickHouse体验平台](#) 允许人们通过即时运行查询来尝试ClickHouse，而无需设置他们的服务器或集群。

体验平台中提供了几个示例数据集以及显示ClickHouse特性的示例查询。

查询以只读用户身份执行。这意味着一些局限性：

- 不允许DDL查询
- 不允许插入查询

还强制执行以下设置：

```

- max_result_bytes=10485760
- max_result_rows=2000
- result_overflow_mode=break
- max_execution_time=60000

```

- max_execution_time=00000

ClickHouse体验还有如下：

[ClickHouse管理服务](#)

实例托管 [Yandex云](#).

更多信息 [云提供商](#).

ClickHouse体验平台界面实际上是通过ClickHouse [HTTP API](#)接口实现的.

体验平台后端只是一个ClickHouse集群，没有任何额外的服务器端应用程序。

体验平台也同样提供了ClickHouse [HTTPS](#)服务端口。

您可以使用任何[HTTP](#)客户端向体验平台进行查询，例如 [curl](#) 或 [wget](#)，或使用以下方式建立连接 [JDBC](#) 或 [ODBC](#) 司机有关支持ClickHouse的软件产品的更多信息，请访问 [这里](#).

参数	值
服务端口	https://play-api.clickhouse.tech:8443
用户	playground
密码	clickhouse

请注意，此服务端口需要安全连接。

示例：

```
curl "https://play-api.clickhouse.tech:8443/?query=SELECT+'Play+ClickHouse!';&user=playground&password=clickhouse&database=datasets"
```

入门

如果您是ClickHouse的新手，并希望亲身体验它的性能，首先您需要通过 [安装过程](#).

之后，您可以选择以下选项之一：

- [通过详细的教程](#)
- [试验示例数据集](#)

安装

系统要求

ClickHouse可以在任何具有x86_64，AArch64或PowerPC64LE CPU架构的Linux，FreeBSD或Mac OS X上运行。

虽然预构建的二进制文件通常是为x86 _ 64编译并利用SSE 4.2指令集，但除非另有说明，否则使用支持它的CPU将成为额外的系统要求。这是检查当前CPU是否支持SSE 4.2的命令：

```
$ grep -q sse4_2 /proc/cpuinfo && echo "SSE 4.2 supported" || echo "SSE 4.2 not supported"
```

要在不支持SSE 4.2或具有AArch64或PowerPC64LE体系结构的处理器上运行ClickHouse，您应该通过源构建ClickHouse进行适当的配置调整。

可用的安装选项

建议为Debian或Ubuntu使用官方的预编译deb软件包。运行以下命令以安装软件包：

然后运行：

```
sudo apt-get install apt-transport-https ca-certificates dirmngr  
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv E0C56BD4  
  
echo "deb https://repo.clickhouse.tech/deb/stable/ main/" | sudo tee \  
    /etc/apt/sources.list.d/clickhouse.list  
sudo apt-get update  
  
sudo apt-get install -y clickhouse-server clickhouse-client  
  
sudo service clickhouse-server start  
clickhouse-client
```

你也可以从这里手动下载安装包：<https://repo.clickhouse.tech/deb/stable/main/>。

如果你想使用最新的测试版本，请使用 `testing` 替换 `stable`。

来自 RPM 包

Yandex ClickHouse 团队建议使用官方预编译的 `rpm` 软件包，用于 CentOS，RedHat 和所有其他基于 `rpm` 的 Linux 发行版。

首先，您需要添加官方存储库：

```
sudo yum install yum-utils  
sudo rpm --import https://repo.yandex.ru/clickhouse/CLICKHOUSE-KEY.GPG  
sudo yum-config-manager --add-repo https://repo.yandex.ru/clickhouse/rpm/stable/x86_64
```

如果您想使用最新版本，请将 `stable` 替换为 `testing`（建议您在测试环境中使用）。

然后运行这些命令以实际安装包：

```
sudo yum install clickhouse-server clickhouse-client
```

您也可以从此处手动下载和安装软件包：https://repo.yandex.ru/clickhouse/rpm/stable/x86_64。

来自 Docker

要在 Docker 中运行 ClickHouse，请遵循 [码头工人中心](#) 上的指南。那些图像使用官方的 `deb` 包。

使用源码安装

具体编译方式可以参考 `build.md`。

你可以编译并安装它们。

你也可以直接使用而不进行安装。

```
Client: programs/clickhouse-client  
Server: programs/clickhouse-server
```

在服务器中为数据创建如下目录：

```
/opt/clickhouse/data/default/  
/opt/clickhouse/metadata/default/
```

（它们可以在 `server config` 中配置。）

为需要的用户运行 `'chown'`

日志的路径可以在 `server config` (`src/programs/server/config.xml`) 中配置。

启动

可以运行如下命令在后台启动服务：

```
sudo service clickhouse-server start
```

可以在 `/var/log/clickhouse-server/` 目录中查看日志。

如果服务没有启动，请检查配置文件 `/etc/clickhouse-server/config.xml`。

你也可以在控制台中直接启动服务：

```
clickhouse-server --config-file=/etc/clickhouse-server/config.xml
```

在这种情况下，日志将被打印到控制台中，这在开发过程中很方便。

如果配置文件在当前目录中，你可以不指定`'-config-file'`参数。它默认使用`'./config.xml'`。

你可以使用命令行客户端连接到服务：

```
clickhouse-client
```

默认情况下它使用`'default'`用户无密码的与`localhost:9000`服务建立连接。

客户端也可以用于连接远程服务，例如：

```
clickhouse-client --host=example.com
```

有关更多信息，请参考«Command-line client»部分。

检查系统是否工作：

```
miloividov@hostname:~/work/metrica/src/src/Client$ ./clickhouse-client
ClickHouse client version 0.0.18749.
Connecting to localhost:9000.
Connected to ClickHouse server version 0.0.18749.
```

```
:) SELECT 1
```

```
SELECT 1
```

```
1
| 1 |

```

```
1 rows in set. Elapsed: 0.003 sec.
```

```
:)
```

恭喜，系统已经工作了！

为了继续进行实验，你可以尝试下载测试数据集。

ClickHouse版本v20.3.4.10,2020-03-20

错误修复

- 此版本还包含20.1.8.41的所有错误修复
- 修复丢失 `rows_before_limit_at_least` 用于通过http进行查询（使用处理器管道）。这修复 #9730. #9757 (尼古拉*科切托夫)

ClickHouse释放v20.3.3.6,2020-03-17

错误修复

- 此版本还包含20.1.7.38的所有错误修复
- 修复复制中的错误，如果用户在以前的版本上执行了突变，则不允许复制工作。这修复 #9645. #9652 (阿利沙平)。它使版本20.3再次向后兼容。
- 添加设置 `use_compact_format_in_distributed_parts_names` 它允许写文件 `INSERT` 查询到 `Distributed` 表格格式更紧凑。这修复 #9647. #9653 (阿利沙平)。它使版本20.3再次向后兼容。

ClickHouse版本v20.3.2.1,2020-03-12

向后不兼容的更改

- 修正了这个问题 `file name too long` 当发送数据 `Distributed` 大量副本的表。修复了服务器日志中显示副本凭据的问题。磁盘上的目录名格式已更改为 `[shard{shard_index}[_replica{replica_index}]]`. #8911 (米哈伊尔*科罗托夫) 升级到新版本后，您将无法在没有人工干预的情况下降级，因为旧的服务器版本无法识别新的目录格式。如果要降级，则必须手动将相应的目录重命名为旧格式。仅当您使用了异步时，此更改才相关 `INSERTS` 到 `Distributed` 桌子在版本20.3.3中，我们将介绍一个设置，让您逐渐启用新格式。
- 更改了mutation命令的复制日志条目的格式。在安装新版本之前，您必须等待旧的突变处理。
- 实现简单的内存分析器，将堆栈跟踪转储到 `system.trace_log` 超过软分配限制的每N个字节 #8765 (伊万) #9472 (阿列克谢-米洛维多夫) 列 `system.trace_log` 从改名 `timer_type` 到 `trace_type`。这将需要改变第三方性能分析和flamegraph处理工具。
- 在任何地方使用操作系统线程id，而不是内部线程编号。这修复 #7477 老 `clickhouse-client` 无法接收从服务器发送的日志，当设置 `send_logs_level` 已启用，因为结构化日志消息的名称和类型已更改。另一方面，不同的服务器版本可以相互发送不同类型的日志。当你不使用 `send_logs_level` 设置，你不应该关心。#8954 (阿列克谢-米洛维多夫)
- 删除 `indexHint` 功能 #9542 (阿列克谢-米洛维多夫)
- 删除 `findClusterIndex`, `findClusterValue` 功能。这修复 #8641. 如果您正在使用这些功能，请发送电子邮件至 `clickhouse-feedback@yandex-team.com` #9543 (阿列克谢-米洛维多夫)
- 现在不允许创建列或添加列 `SELECT` 子查询作为默认表达式。#9481 (阿利沙平)
- 需要联接中的子查询的别名。#9274 (Artem Zuikov)
- 改进 `ALTER MODIFY/ADD` 查询逻辑。现在你不能 `ADD` 不带类型的列, `MODIFY` 默认表达式不改变列的类型和 `MODIFY type` 不会丢失默认表达式值。修复 #8669. #9227 (阿利沙平)
- 要求重新启动服务器以应用日志记录配置中的更改。这是一种临时解决方法，可以避免服务器将日志记录到已删除的日志文件中的错误（请参阅 #8696). #8707 (Alexander Kuzmenkov)
- 设置 `experimental_use_processors` 默认情况下启用。此设置允许使用新的查询管道。这是内部重构，我们期望没有明显的变化。如果您将看到任何问题，请将其设置为返回零。#8768 (阿列克谢-米洛维多夫)

新功能

- 添加 `Avro` 和 `AvroConfluent` 输入/输出格式 #8571 (安德鲁Onyshchuk) #8957 (安德鲁Onyshchuk) #8717 (阿列克谢-米洛维多夫)
- 过期密钥的多线程和非阻塞更新 `cache` 字典（可选的权限读取旧的）。#8303 (尼基塔*米哈伊洛夫)
- 添加查询 `ALTER ... MATERIALIZE TTL`。它运行突变，强制通过TTL删除过期的数据，并重新计算所有部分有关ttl的元信息。#8775 (安东*波波夫)
- 如果需要，从HashJoin切换到MergeJoin（在磁盘上 #9082 (Artem Zuikov)
- 已添加 `MOVE PARTITION` 命令 `ALTER TABLE #4729 #6168` (纪尧姆*塔瑟里)
- 动态地从配置文件重新加载存储配置。#8594 (Vladimir Chebotarev)
- 允许更改 `storage_policy` 为了不那么富有的人。#8107 (Vladimir Chebotarev)
- 增加了对s3存储和表功能的glob支持。#8851 (Vladimir Chebotarev)
- 执行 `bitAnd`, `bitOr`, `bitXor`, `bitNot` 为 `FixedString(N)` 数据类型。#9091 (纪尧姆*塔瑟里)
- 添加功能 `bitCount`。这修复 #8702. #8708 (阿列克谢-米洛维多夫) #8749 (ikonvlov)

- 添加 `generateRandom` 表函数生成具有给定模式的随机行。允许用数据填充任意测试表。#8994 (Ilya Yatsishin)
- `JSONEachRowFormat`：当对象包含在顶层数组中时，支持特殊情况。#8860 (克鲁格洛夫*帕维尔)
- 现在可以创建一个列 `DEFAULT` 取决于默认列的表达式 `ALIAS` 表达。#9489 (阿利沙平)
- 允许指定 `--limit` 超过源数据大小 `clickhouse-obfuscator`. 数据将以不同的随机种子重复。#9155 (阿列克谢-米洛维多夫)
- 已添加 `groupArraySample` 功能（类似于 `groupArray`）与 `reservior` 采样算法。#8286 (阿莫斯鸟)
- 现在，您可以监视更新队列的大小 `cache/complex_key_cache` 通过系统指标字典。#9413 (尼基塔*米哈伊洛夫)
- 允许使用 CRLF 作为 CSV 输出格式的行分隔符与设置 `output_format_csv_crlf_end_of_line` 设置为 1 #8934 #8935 #8963 (米哈伊尔*科罗托夫)
- 实现的更多功能 H3 API: `h3GetBaseCell`, `h3HexAreaM2`, `h3IndexesAreNeighbors`, `h3ToChildren`, `h3ToString` 和 `stringToH3` #8938 (Nico Mandery)
- 引入新设置: `max_parser_depth` 控制最大堆栈大小并允许大型复杂查询。这修复 #6681 和 #7668. #8647 (马克西姆*斯米尔诺夫)
- 添加设置 `force_optimize_skip_unused_shards` 如果无法跳过未使用的分片，则设置为抛出 #8805 (Azat Khuzhin)
- 允许配置多个磁盘/卷用于存储数据发送 `Distributed` 发动机 #8756 (Azat Khuzhin)
- 支持存储策略 (`<tmp_policy>`) 用于存储临时数据。#8750 (Azat Khuzhin)
- 已添加 `X-ClickHouse-Exception-Code` 如果在发送数据之前引发异常，则设置的 HTTP 头。这实现了 #4971. #8786 (米哈伊尔*科罗托夫)
- 添加功能 `ifNotFinite`. 这只是一个句法糖: `ifNotFinite(x, y) = isFinite(x) ? x : y.` #8710 (阿列克谢-米洛维多夫)
- 已添加 `last_successful_update_time` 列中 `system.dictionaries` 表 #9394 (尼基塔*米哈伊洛夫)
- 添加 `blockSerializedSize` 功能（磁盘大小不压缩）#8952 (Azat Khuzhin)
- 添加功能 `moduloOrZero` #9358 (hcz)
- 添加系统表 `system.zeros` 和 `system.zeros_mt` 以及故事功能 `zeros()` 和 `zeros_mt()`. 表（和表函数）包含具有名称的单列 `zero` 和类型 `UInt8`. 此列包含零。为了测试目的，需要它作为生成许多行的最快方法。这修复 #6604 #9593 (尼古拉*科切托夫)

实验特点

- 添加新的紧凑格式的部件 `MergeTree`-家庭表中的所有列都存储在一个文件中。它有助于提高小型和频繁插入的性能。旧的格式（每列一个文件）现在被称为 `wide`。数据存储格式由设置控制 `min_bytes_for_wide_part` 和 `min_rows_for_wide_part`. #8290 (安东*波波夫)
- 支持 S3 存储 `Log`, `TinyLog` 和 `StripeLog` 桌子 #8862 (帕维尔*科瓦连科)

错误修复

- 修正了日志消息中不一致的空格。#9322 (阿列克谢-米洛维多夫)
- 修复在创建表时将未命名元组数组展平为嵌套结构的错误。#8866 (achulkov2)
- 修复了以下问题“Too many open files”如果有太多的文件匹配 glob 模式可能会发生错误 `File` 表或 `file` 表功能。现在文件懒洋洋地打开。这修复 #8857 #8861 (阿列克谢-米洛维多夫)
- 删除临时表现在只删除临时表。#8907 (维塔利*巴拉诺夫)
- 当我们关闭服务器或分离/附加表时删除过时的分区。#8602 (纪尧姆*塔瑟里)
- 默认磁盘如何计算可用空间 `data` 子目录。修复了可用空间量计算不正确的问题，如果 `data` 目录被安装到一个单独的设备（罕见的情况）。这修复 #7441 #9257 (米哈伊尔*科罗托夫)
- 允许逗号（交叉）与 IN () 内部连接。#9251 (Artem Zuikov)
- 如果在 WHERE 部分中有 [NOT] LIKE 运算符，则允许将 CROSS 重写为 INNER JOIN。#9229 (Artem Zuikov)
- 修复后可能不正确的结果 `GROUP BY` 启用设置 `distributed_aggregation_memory_efficient`. 修复 #9134. #9289 (尼古拉*科切托夫)
- 找到的键在缓存字典的指标中被计为错过。#9411 (尼基塔*米哈伊洛夫)
- 修复引入的复制协议不兼容 #8598. #9412 (阿利沙平)
- 在固定的竞争条件 `queue_task_handle` 在启动 `ReplicatedMergeTree` 桌子 #9552 (阿列克谢-米洛维多夫)
- 令牌 NOT 没有工作 `SHOW TABLES NOT LIKE` 查询 #8727 #8940 (阿列克谢-米洛维多夫)
- 添加范围检查功能 `h3EdgeLengthM`. 如果没有这个检查，缓冲区溢出是可能的。#8945 (阿列克谢-米洛维多夫)
- 修复了多个参数（超过 10）的三元逻辑运算批量计算中的错误。#8718 (亚历山大*卡扎科夫)
- 修复 PREWHERE 优化的错误，这可能导致段错误或 `Inconsistent number of columns got from MergeTreeRangeReader` 例外。#9024 (安东*波波夫)
- 修复意外 `Timeout exceeded while reading from socket` 异常，在实际超时之前以及启用查询探查器时，在安全连接上

- 随机发生。还添加 `connect_timeout_with_failover_secure_ms` 设置（默认100ms），这是类似于 `connect_timeout_with_failover_ms`，但用于安全连接（因为SSL握手比普通TCP连接慢）#9026 (tavplubix)
- 修复突变最终确定的错误，当突变可能处于以下状态时 `parts_to_do=0` 和 `is_done=0`. #9022 (阿利沙平)
 - 使用新的任何连接逻辑 `partial_merge_join` 设置。有可能使 ANY|ALL|SEMI LEFT 和 ALL INNER 加入与 `partial_merge_join=1` 现在 #8932 (Artem Zuikov)
 - Shard现在将从发起者获得的设置夹到shard的constraints，而不是抛出异常。此修补程序允许将查询发送到具有另一个约束的分片。#9447 (维塔利*巴拉诺夫)
 - 修正了内存管理问题 `MergeTreeReadPool`. #8791 (Vladimir Chebotarev)
 - 修复 `toDecimal*OrNull()` 使用字符串调用时的函数系列 e. 修复 #8312 #8764 (Artem Zuikov)
 - 请确保 `FORMAT Null` 不向客户端发送数据。#8767 (Alexander Kuzmenkov)
 - 修复时间戳中的错误 `LiveViewBlockInputStream` 不会更新。`LIVE VIEW` 是一个实验特征。#8644 (vxider) #8625 (vxider)
 - 固定 `ALTER MODIFY TTL` 不允许删除旧ttl表达式的错误行为。#8422 (Vladimir Chebotarev)
 - 修复了MergeTreeIndexSet中的UBSan报告。这修复 #9250 #9365 (阿列克谢-米洛维多夫)
 - 固定的行为 `match` 和 `extract` 当干草堆有零字节的函数。当干草堆不变时，这种行为是错误的。这修复 #9160 #9163 (阿列克谢-米洛维多夫) #9345 (阿列克谢-米洛维多夫)
 - 避免从apache Avro第三方库中的析构函数抛出。#9066 (安德鲁Onyshchuk)
 - 不要提交从轮询的批次 `Kafka` 部分，因为它可能会导致数据漏洞。#8876 (filimonov)
 - 修复 `joinGet` 使用可为空的返回类型。<https://github.com/ClickHouse/ClickHouse/issues/8919> #9014 (阿莫斯鸟)
 - 修复压缩时的数据不兼容 `T64` 编解ec #9016 (Artem Zuikov) 修复数据类型id `T64` 在受影响的版本中导致错误(de)压缩的压缩编解ec。#9033 (Artem Zuikov)
 - 添加设置 `enable_early_constant_folding` 并禁用它在某些情况下，导致错误。#9010 (Artem Zuikov)
 - 使用VIEW修复下推谓词优化器并启用测试 #9011 (张冬)
 - 修复段错误 `Merge` 表，从读取时可能发生 `File` 储存 #9387 (tavplubix)
 - 添加了对存储策略的检查 `ATTACH PARTITION FROM`, `REPLACE PARTITION`, `MOVE TO TABLE`. 否则，它可以使部分数据重新启动后无法访问，并阻止ClickHouse启动。#9383 (Vladimir Chebotarev)
 - 修复改变，如果有TTL设置表。#8800 (安东*波波夫)
 - 修复在以下情况下可能发生的竞争条件 `SYSTEM RELOAD ALL DICTIONARIES` 在某些字典被修改/添加/删除时执行。#8801 (维塔利*巴拉诺夫)
 - 在以前的版本 `Memory` 数据库引擎使用空数据路径，因此在以下位置创建表 `path directory` (e.g. `/var/lib/clickhouse/`), not in data directory of database (e.g. `/var/lib/clickhouse/db_name`). #8753 (tavplubix)
 - 修复了关于缺少默认磁盘或策略的错误日志消息。#9530 (Vladimir Chebotarev)
 - 修复数组类型的**bloom_filter**索引的not(has())。#9407 (achimbab)
 - 允许表中的第一列 `Log` 引擎是别名 #9231 (伊万)
 - 从读取时修复范围的顺序 `MergeTree` 表中的一个线程。它可能会导致例外 `MergeTreeRangeReader` 或错误的查询结果。#9050 (安东*波波夫)
 - 修复眉露>> `reinterpretAsFixedString` 返回 `FixedSize` 而不是 `String`. #9052 (安德鲁Onyshchuk)
 - 避免极少数情况下，当用户可以得到错误的错误消息 (`Success` 而不是详细的错误描述)。#9457 (阿列克谢-米洛维多夫)
 - 使用时不要崩溃 `Template` 使用空行模板格式化。#8785 (Alexander Kuzmenkov)
 - 系统表的元数据文件可能在错误的位置创建 #8653 (tavplubix) 修复 #8581.
 - 修复缓存字典中exception_ptr上的数据竞赛 #8303. #9379 (尼基塔*米哈伊洛夫)
 - 不要为查询引发异常 `ATTACH TABLE IF NOT EXISTS`. 以前它是抛出，如果表已经存在，尽管 `IF NOT EXISTS` 条款 #8967 (安东*波波夫)
 - 修复了异常消息中丢失的关闭paren。#8811 (阿列克谢-米洛维多夫)
 - 避免消息 `Possible deadlock avoided` 在clickhouse客户端在交互模式下启动。#9455 (阿列克谢-米洛维多夫)
 - 修复了base64编码值末尾填充格式错误的问题。更新base64库。这修复 #9491，关闭 #9492 #9500 (阿列克谢-米洛维多夫)
 - 防止丢失数据 `Kafka` 在极少数情况下，在读取后缀之后但在提交之前发生异常。修复 #9378 #9507 (filimonov)
 - 在固定的异常 `DROP TABLE IF EXISTS` #8663 (尼基塔*瓦西列夫)
 - 修复当用户尝试崩溃 `ALTER MODIFY SETTING` 对于老格式化 `MergeTree` 表引擎家族. #9435 (阿利沙平)
 - 支持在JSON相关函数中不适合Int64的UInt64号码。更新SIMDJSON掌握。这修复 #9209 #9344 (阿列克谢-米洛维多夫)

^ ^,

- 当使用非严格单调函数索引时，固定执行反转谓词。 #9223 (亚历山大*卡扎科夫)
- 不要试图折叠 IN 常量在 GROUP BY #8868 (阿莫斯鸟)
- 修复bug ALTER DELETE 突变导致索引损坏。这修复 #9019 和 #8982. 另外修复极其罕见的竞争条件 ReplicatedMergeTree ALTER 查询。 #9048 (阿利沙平)
- 当设置 compile_expressions 被启用，你可以得到 unexpected column 在 LLVMExecutableFunction 当我们使用 Nullable 类型 #8910 (纪尧姆*塔瑟里)
- 多个修复 Kafka 引擎：1) 修复在消费者组重新平衡期间出现的重复项。2) 修复罕见 ‘holes’ 当数据从一个轮询的几个分区轮询并部分提交时出现（现在我们总是处理/提交整个轮询的消息块）。3) 通过块大小修复刷新（在此之前，只有超时刷新才能正常工作）。4) 更好的订阅程序（与分配反馈）。5) 使测试工作得更快（默认时间间隔和超时）。由于数据之前没有被块大小刷新（根据文档），pr 可能会导致默认设置的性能下降（由于更频繁和更小的刷新不太理想）。如果您在更改后遇到性能问题-请增加 kafka_max_block_size 在表中的更大的值（例如 CREATE TABLE ...Engine=Kafka ... SETTINGS kafka_max_block_size=524288). 修复 #7259 #8917 (filimonov)
- 修复 Parameter out of bound 在 PREWHERE 优化之后的某些查询中出现异常。 #8914 (Baudouin Giard)
- 修正了函数参数混合常量的情况 arrayZip. #8705 (阿列克谢-米洛维多夫)
- 执行时 CREATE 查询，在存储引擎参数中折叠常量表达式。将空数据库名称替换为当前数据库。修复 #6508, #3492 #9262 (tavplubix)
- 现在不可能创建或添加具有简单循环别名的列，如 a DEFAULT b, b DEFAULT a. #9603 (阿利沙平)
- 修正了双重移动可能会损坏原始部分的错误。这是相关的，如果你使用 ALTER TABLE MOVE #8680 (Vladimir Chebotarev)
- 允许 interval 用于正确解析的标识符，而无需反引号。当一个查询不能被执行，即使固定的问题 interval 标识符用反引号或双引号括起来。这修复 #9124. #9142 (阿列克谢-米洛维多夫)
- 修正了模糊测试和不正确的行为 bitTestAll/bitTestAny 功能。 #9143 (阿列克谢-米洛维多夫)
- 修复可能的崩溃/错误的行数 LIMIT n WITH TIES 当有很多行等于第n行时。 #9464 (tavplubix)
- 使用enabled编写的部件修复突变 insert_quorum. #9463 (阿利沙平)
- 修复数据竞赛破坏 Poco::HTTPServer. 当服务器启动并立即关闭时，可能会发生这种情况。 #9468 (安东*波波夫)
- 修复运行时显示误导性错误消息的错误 SHOW CREATE TABLE a_table_that_does_not_exist. #8899 (achulkov2)
- 固定 Parameters are out of bound 例外在一些罕见的情况下，当我们在一个常数 SELECT 条款时，我们有一个 ORDER BY 和一个 LIMIT 条款 #8892 (纪尧姆*塔瑟里)
- 修复突变定稿，当已经完成突变可以有状态 is_done=0. #9217 (阿利沙平)
- 防止执行 ALTER ADD INDEX 对于旧语法的MergeTree表，因为它不起作用。 #8822 (米哈伊尔*科罗托夫)
- 在服务器启动时不要访问表，这 LIVE VIEW 取决于，所以服务器将能够启动。也删除 LIVE VIEW 分离时的依赖关系 LIVE VIEW. LIVE VIEW 是一个实验特征。 #8824 (tavplubix)
- 修复可能的段错误 MergeTreeRangeReader，同时执行 PREWHERE. #9106 (安东*波波夫)
- 修复与列TTL可能不匹配的校验和。 #9451 (安东*波波夫)
- 修正了一个错误，当部分没有被移动的情况下，只有一个卷的TTL规则在后台。 #8672 (Vladimir Chebotarev)
- 修正了这个问题 Method createColumn() is not implemented for data type Set. 这修复 #7799. #8674 (阿列克谢-米洛维多夫)
- 现在我们将尝试更频繁地完成突变。 #9427 (阿利沙平)
- 修复 intDiv 减一个常数 #9351 (hcj)
- 修复可能的竞争条件 BlockIO. #9356 (尼古拉*科切托夫)
- 修复尝试使用/删除时导致服务器终止的错误 Kafka 使用错误的参数创建的表。 #9513 (filimonov)
- 增加了解决方法，如果操作系统返回错误的结果 timer_create 功能。 #8837 (阿列克谢-米洛维多夫)
- 在使用固定错误 min_marks_for_seek 参数。修复了分布式表中没有分片键时的错误消息，并且我们尝试跳过未使用的分片。 #8908 (Azat Khuzhin)

改进

- 执行 ALTER MODIFY/DROP 对突变的顶部查询 ReplicatedMergeTree* 引擎家族. 现在 ALTERS 仅在元数据更新阶段阻止，之后不阻止。 #8701 (阿利沙平)
- 添加重写交叉到内部连接的能力 WHERE 包含未编译名称的部分。 #9512 (Artem Zuikov)
- 赖眉露>> SHOW TABLES 和 SHOW DATABASES 查询支持 WHERE 表达式和 FROM/IN #9076 (sundyli)
- 添加了一个设置 deduplicate_blocks_in_dependent_materialized_views. #9070 (urykhy)
- 在最近的变化之后，MySQL客户端开始以十六进制打印二进制字符串，从而使它们不可读 (#9032). ClickHouse中的解决方法是将字符串列标记为UTF-8，这并不总是如此，但通常是这种情况。 #9079 (尤里*巴拉诺夫)
- 添加对字符串和FixedString键的支持 sumMap #8903 (Baudouin Giard)

- 支持SummingMergeTree地图中的字符串键 #8933 (Baudouin Giard)
- 即使线程已抛出异常，也向线程池发送线程终止信号 #8736 (丁香飞)
- 允许设置 `query_id` 在 `clickhouse-benchmark` #9416 (安东*波波夫)
- 不要让奇怪的表达 `ALTER TABLE ... PARTITION partition` 查询。这个地址 #7192 #8835 (阿列克谢-米洛维多夫)
- 表 `system.table_engines` 现在提供有关功能支持的信息 (如 `supports_ttl` 或 `supports_sort_order`). #8830 (Max Akhmedov)
- 启用 `system.metric_log` 默认情况下。它将包含具有ProfileEvents值的行，CurrentMetrics收集与“collect_interval_milliseconds”间隔（默认情况下为一秒）。该表非常小（通常以兆字节为单位），默认情况下收集此数据是合理的。#9225 (阿列克谢-米洛维多夫)
- Initialize query profiler for all threads in a group, e.g. it allows to fully profile insert-queries. Fixes #6964 #8874 (伊万)
- 现在是暂时的 `LIVE VIEW` 创建者 `CREATE LIVE VIEW name WITH TIMEOUT [42] ...` 而不是 `CREATE TEMPORARY LIVE VIEW ...`，因为以前的语法不符合 `CREATE TEMPORARY TABLE ...` #9131 (tavplubix)
- 添加`text_log`。级别配置参数，以限制进入 `system.text_log` 表 #8809 (Azat Khuzhin)
- 允许根据TTL规则将下载的部分放入磁盘/卷 #8598 (Vladimir Chebotarev)
- 对于外部MySQL字典，允许将MySQL连接池共同化为“share”他们在字典中。此选项显着减少到MySQL服务器的连接数。#9409 (Clément Rodriguez)
- 显示分位数的最近查询执行时间 `clickhouse-benchmark` 输出而不是插值值。最好显示与某些查询的执行时间相对应的值。#8712 (阿列克谢-米洛维多夫)
- 可以在将数据插入到Kafka时为消息添加密钥和时间戳。修复 #7198 #8969 (filimonov)
- 如果服务器从终端运行，请按颜色突出显示线程号，查询id和日志优先级。这是为了提高开发人员相关日志消息的可读性。#8961 (阿列克谢-米洛维多夫)
- 更好的异常消息，同时加载表 `Ordinary` 数据库。#9527 (阿列克谢-米洛维多夫)
- 执行 `arraySlice` 对于具有聚合函数状态的数组。这修复 #9388 #9391 (阿列克谢-米洛维多夫)
- 允许在`in`运算符的右侧使用常量函数和常量数组。#8813 (安东*波波夫)
- 如果在获取系统数据时发生了zookeeper异常。副本，将其显示在单独的列中。这实现了 #9137 #9138 (阿列克谢-米洛维多夫)
- 原子删除`destroy`上的MergeTree数据部分。#8402 (Vladimir Chebotarev)
- 支持分布式表的行级安全性。#8926 (伊万)
- Now we recognize suffix (like KB, KiB...) in settings values. #8072 (米哈伊尔*科罗托夫)
- 在构建大型连接的结果时防止内存不足。#8637 (Artem Zuikov)
- 在交互模式下为建议添加群集名称 `clickhouse-client`. #8709 (阿列克谢-米洛维多夫)
- Initialize query profiler for all threads in a group, e.g. it allows to fully profile insert-queries #8820 (伊万)
- 添加列 `exception_code` 在 `system.query_log` 桌子 #8770 (米哈伊尔*科罗托夫)
- 在端口上启用MySQL兼容服务器 9004 在默认服务器配置文件中。在配置的例子固定密码生成命令。#8771 (尤里*巴拉诺夫)
- 如果文件系统是只读的，请防止在关闭时中止。这修复 #9094 #9100 (阿列克谢-米洛维多夫)
- 当HTTP POST查询中需要长度时，更好的异常消息。#9453 (阿列克谢-米洛维多夫)
- 添加 `_path` 和 `_file` 虚拟列 `HDFS` 和 `File` 发动机和 `hdfs` 和 `file` 表函数 #8489 (Olga Khvostikova)
- 修复错误 `Cannot find column` 同时插入到 `MATERIALIZED VIEW` 在情况下，如果新列被添加到视图的内部表。#8766 #8788 (vzakaznikov) #8788 #8806 (尼古拉*科切托夫) #8803 (尼古拉*科切托夫)
- 通过最终更新后发送进度（如日志）修复本机客户端-服务器协议的进度。这可能仅与使用本机协议的某些第三方工具相关。#9495 (Azat Khuzhin)
- 添加系统指标跟踪使用MySQL协议的客户端连接数 (#9013). #9015 (尤金*克里莫夫)
- 从现在开始，HTTP响应将有 `X-ClickHouse-Timezone` 标题设置为相同的时区值 `SELECT timezone()` 会报告。#9493 (Denis Glazachev)

性能改进

- 使用IN提高分析指标的性能 #9261 (安东*波波夫)
- 逻辑函数+代码清理更简单，更有效的代码。跟进到 #8718 #8728 (亚历山大*卡扎科夫)
- 整体性能改善（范围为5%。通过确保使用C++20功能进行更严格的别名处理，对于受影响的查询来说，这是200%）。#9304 (阿莫斯鸟)
- 比较函数的内部循环更严格的别名。#9327 (阿列克谢-米洛维多夫)
- 对于算术函数的内部循环更严格的别名。#9325 (阿列克谢-米洛维多夫)
- `ColumnVector::replicate()`的实现速度快约3倍，通过该实现`ColumnConst::convertToFullColumn()`。在实现常

数时，也将在测试中有用。 #9293 (亚历山大*卡扎科夫)

- 另一个小小的性能改进 `ColumnVector::replicate()` (这加快了 `materialize` 函数和高阶函数) , 甚至进一步改进 #9293 #9442 (亚历山大*卡扎科夫)
- 改进的性能 `stochasticLinearRegression` 聚合函数。此补丁由英特尔贡献。#8652 (阿列克谢-米洛维多夫)
- 提高性能 `reinterpretAsFixedString` 功能。#9342 (阿列克谢-米洛维多夫)
- 不要向客户端发送块 `Null` 处理器管道中的格式。#8797 (尼古拉*科切托夫) #8767 (Alexander Kuzmenkov)

构建/测试/包装改进

- 异常处理现在可以在适用于Linux的Windows子系统上正常工作。看<https://github.com/ClickHouse-Extras/libunwind/pull/3> 这修复 #6480 #9564 (sobolevsv)
- 替换 `readline` 与 `replxx` 对于在交互式线编辑 `clickhouse-client` #8416 (伊万)
- 在 `FunctionsComparison` 中更好的构建时间和更少的模板实例化。#9324 (阿列克谢-米洛维多夫)
- 增加了与集成 `clang-tidy` 在线人 另请参阅 #6044 #9566 (阿列克谢-米洛维多夫)
- 现在我们使用CI链接ClickHouse `lld` 即使是 `gcc`. #9049 (阿利沙平)
- 允许随机线程调度和插入毛刺时 `THREAD_FUZZER_*` 设置环境变量。这有助于测试。#9459 (阿列克谢-米洛维多夫)
- 在无状态测试中启用安全套接字 #9288 (tavplubix)
- 使 `SPLIT_SHARED_LIBRARIES=OFF` 更强大 #9156 (Azat Khuzhin)
- 眉眉露>> “`performance_introspection_and_logging`” 测试可靠的随机服务器卡住。这可能发生在CI环境中。另请参阅 #9515 #9528 (阿列克谢-米洛维多夫)
- 在样式检查中验证XML。#9550 (阿列克谢-米洛维多夫)
- 修正了测试中的竞争条件 `00738_lock_for_inner_table`. 这个测试依赖于睡眠。#9555 (阿列克谢-米洛维多夫)
- 删除类型的性能测试 `once`. 这是在统计比较模式下运行所有性能测试（更可靠）所需的。#9557 (阿列克谢-米洛维多夫)
- 增加了算术函数的性能测试。#9326 (阿列克谢-米洛维多夫)
- 增加了性能测试 `sumMap` 和 `sumMapWithOverflow` 聚合函数。后续行动 #8933 #8947 (阿列克谢-米洛维多夫)
- 通过样式检查确保错误代码的样式。#9370 (阿列克谢-米洛维多夫)
- 为测试历史添加脚本。#8796 (阿利沙平)
- 添加GCC警告 `-Wsuggest-override` 找到并修复所有地方 `override` 必须使用关键字。#8760 (kreuzerkrieg)
- 在Mac OS X下忽略弱符号，因为它必须被定义 #9538 (已删除用户)
- 规范性能测试中某些查询的运行时间。这是在准备在比较模式下运行所有性能测试时完成的。#9565 (阿列克谢-米洛维多夫)
- 修复一些测试，以支持pytest与查询测试 #9062 (伊万)
- 使用MSan在生成中启用SSL，因此在运行无状态测试时，服务器不会在启动时失败 #9531 (tavplubix)
- 修复测试结果中的数据库替换 #9384 (Ilya Yatsishin)
- 针对其他平台构建修复程序 #9381 (proller) #8755 (proller) #8631 (proller)
- 将磁盘部分添加到无状态复盖率测试docker映像 #9213 (帕维尔*科瓦连科)
- 使用GRPC构建时，摆脱源代码树中的文件 #9588 (阿莫斯鸟)
- 通过从上下文中删除SessionCleaner来缩短构建时间。让SessionCleaner的代码更简单。#9232 (阿列克谢-米洛维多夫)
- 更新了clickhouse-test脚本中挂起查询的检查 #8858 (亚历山大*卡扎科夫)
- 从存储库中删除了一些无用的文件。#8843 (阿列克谢-米洛维多夫)
- 更改类型的数学perftests从 `once` 到 `loop`. #8783 (尼古拉*科切托夫)
- 添加码头镜像，它允许为我们的代码库构建交互式代码浏览器HTML报告。#8781 (阿利沙平) 见 Woboq 代码浏览器
- 抑制MSan下的一些测试失败。#8780 (Alexander Kuzmenkov)
- 加速“exception while insert” 测试 此测试通常在具有复盖率的调试版本中超时。#8711 (阿列克谢-米洛维多夫)
- 更新 `libcxx` 和 `libcxxabi` 为了主人在准备 #9304 #9308 (阿列克谢-米洛维多夫)
- 修复flacky测试 `00910_zookeeper_test_alter_compression_codecs`. #9525 (阿列克谢-米洛维多夫)
- 清理重复的链接器标志。确保链接器不会查找意想不到的符号。#9433 (阿莫斯鸟)
- 添加 `clickhouse-odbc` 驱动程序进入测试图像。这允许通过自己的ODBC驱动程序测试ClickHouse与ClickHouse的交互。#9348 (fillimonov)
- 修复单元测试中的几个错误。#9047 (阿利沙平)
- 启用 `-Wmissing-include-dirs` GCC警告消除所有不存在的包括-主要是由于CMake脚本错误 #8704 (kreuzerkrieg)
- 描述查询探查器无法工作的原因。这是用于 #9049 #9144 (阿列克谢-米洛维多夫)
- 将OpenSSL更新到上游主机。修复了TLS连接可能会失败并显示消息的问题 OpenSSL SSL_read: error:14094438:SSL

routines:ssl3_read_bytes:tlsv1 alert internal error 和 SSL Exception: error:2400006E:random number generator::error retrieving entropy. 该问题出现在版本20.1中。 #8956 (阿列克谢-米洛维多夫)

- 更新服务器的Dockerfile #8893 (Ilya Mazaev)
- Build-gcc-from-sources脚本中的小修复 #8774 (Michael Nacharov)
- 替换 numbers 到 zeros 在perftests其中 number 不使用列。这将导致更干净的测试结果。#9600 (尼古拉*科切托夫)
- 修复列构造函数中使用initializer_list时堆栈溢出问题。#9367 (已删除用户)
- 将librdkafka升级到v1.3.0。启用bund绑 rdakafka 和 gsasl mac OS X上的库 #9000 (安德鲁Onyshchuk)
- 在GCC9.2.0上构建修复程序 #9306 (vxider)

碌莽禄.拢.0755-88888888

ClickHouse版本v20.1.8.41,2020-03-20

错误修复

- 修复可能的永久性 Cannot schedule a task 错误（由于未处理的异常 ParallelAggregatingBlockInputStream::Handler::onFinish/onFinishThread）。这修复 #6833. #9154 (Azat Khuzhin)
- 修复过多的内存消耗 ALTER 查询（突变）。这修复 #9533 和 #9670. #9754 (阿利沙平)
- 修复外部字典DDL中反引用的错误。这修复 #9619. #9734 (阿利沙平)

ClickHouse释放v20.1.7.38,2020-03-18

错误修复

- 修正了不正确的内部函数名称 sumKahan 和 sumWithOverflow. 在远程查询中使用此函数时，我会导致异常。#9636 (Azat Khuzhin). 这个问题是在所有ClickHouse版本。
- 允许 ALTER ON CLUSTER 的 Distributed 具有内部复制的表。这修复 #3268. #9617 (shinoi2). 这个问题是在所有ClickHouse版本。
- 修复可能的异常 Size of filter doesn't match size of column 和 Invalid number of rows in Chunk 在 MergeTreeRangeReader. 它们可能在执行时出现 PREWHERE 在某些情况下。修复 #9132. #9612 (安东*波波夫)
- 修复了这个问题：如果你编写一个简单的算术表达式，则不会保留时区 time + 1 （与像这样的表达形成对比 time + INTERVAL 1 SECOND）。这修复 #5743. #9323 (阿列克谢-米洛维多夫). 这个问题是在所有ClickHouse版本。
- 现在不可能创建或添加具有简单循环别名的列，如 a DEFAULT b, b DEFAULT a. #9603 (阿利沙平)
- 修复了base64编码值末尾填充格式错误的问题。更新base64库。这修复 #9491，关闭 #9492 #9500 (阿列克谢-米洛维多夫)
- 修复数据竞赛破坏 Poco::HTTPServer. 当服务器启动并立即关闭时，可能会发生这种情况。#9468 (安东*波波夫)
- 修复可能的崩溃/错误的行数 LIMIT n WITH TIES 当有很多行等于第n行时。#9464 (tavplubix)
- 修复与列TTL可能不匹配的校验和。#9451 (安东*波波夫)
- 修复当用户尝试崩溃 ALTER MODIFY SETTING 对于老格式化 MergeTree 表引擎家族. #9435 (阿利沙平)
- 现在我们将尝试更频繁地完成突变。#9427 (阿利沙平)
- 修复引入的复制协议不兼容 #8598. #9412 (阿利沙平)
- 修复数组类型的bloom_filter索引的not(has())。#9407 (achimbab)
- 固定的行为 match 和 extract 当干草堆有零字节的函数。当干草堆不变时，这种行为是错误的。这修复 #9160 #9163 (阿列克谢-米洛维多夫) #9345 (阿列克谢-米洛维多夫)

构建/测试/包装改进

- 异常处理现在可以在适用于Linux的Windows子系统上正常工作。看<https://github.com/ClickHouse-Extras/libunwind/pull/3> 这修复 #6480 #9564 (sobolevsv)

ClickHouse释放v20.1.6.30,2020-03-05

错误修复

- 修复压缩时的数据不兼容 T64 编解ec #9039 (abyss7)
- 在一个线程中从MergeTree表中读取时修复范围顺序。修复 #8964. #9050 (Curtiz))
- 修复可能的段错误 MergeTreeRangeReader，同时执行 PREWHERE. 修复 #9064. #9106 (Curtiz))

#9100 (Curuzju)

- 修复 `reinterpretAsFixedString` 返回 `FixedString` 而不是 `String`.
#9052 (oandrew)
- 修复 `joinGet` 使用可为空的返回类型。修复 **#8919**
#9014 (amosbird)
- 修复 `bittestall/bitTestAny` 函数的模糊测试和不正确的行为。
#9143 (阿列克谢-米洛维多夫)
- 修复当干草堆有零字节时匹配和提取函数的行为。当干草堆不变时，这种行为是错误的。修复 **#9160**
#9163 (阿列克谢-米洛维多夫)
- 当使用非严格单调函数索引时，固定执行反转谓词。修复 **#9034**
#9223 (Akazz)
- 允许重写 `CROSS` 到 `INNER JOIN` 如果有 `[NOT] LIKE` 操作员在 `WHERE` 科。修复 **#9191**
#9229 (4ertus2)
- 允许使用日志引擎的表中的第一列成为别名。
#9231 (abyss7)
- 允许逗号加入 `IN()` 进去 修复 **#7314**.
#9251 (4ertus2)
- 改进 `ALTER MODIFY/ADD` 查询逻辑。现在你不能 `ADD` 不带类型的列, `MODIFY` 默认表达式不改变列的类型和 `MODIFY type` 不会丢失默认表达式值。修复 **#8669**.
#9227 (alesapin)
- 修复突变最终确定，当已经完成突变时可以具有状态 `is_done=0`。
#9217 (alesapin)
- 碌莽禄Support: “Processors” 管道系统.数字和系统.`numbers_mt` 这也修复了错误时 `max_execution_time` 不被尊重。
#7796 (KochetovNicolai)
- 修复错误的计数 `DictCacheKeysRequestedFound` 公制。
#9411 (nikitamikhaylov)
- 添加了对存储策略的检查 `ATTACH PARTITION FROM`, `REPLACE PARTITION`, `MOVE TO TABLE` 否则可能使部分数据在重新启动后无法访问，并阻止ClickHouse启动。
#9383 (excitoon)
- 在固定的瑞银报告 `MergeTreeIndexSet`. 这修复 **#9250**
#9365 (阿列克谢-米洛维多夫)
- 在 `BlockIO` 中修复可能的数据集。
#9356 (KochetovNicolai)
- 支持 `UInt64` 在 JSON 相关函数中不适合 `Int64` 的数字。更新 `SIMDJSON` 为了主人 这修复 **#9209**
#9344 (阿列克谢-米洛维多夫)
- 如果将数据目录挂载到单独的设备，则修复可用空间量计算不正确时的问题。对于默认磁盘，计算数据子目录的可用空间。这修复 **#7441**
#9257 (米尔布)
- 修复 TLS 连接可能会失败并显示消息时的问题 `OpenSSL SSL_read: error:14094438:SSL routines:ssl3_read_bytes:tlsv1 alert internal error and SSL Exception: error:2400006E:random number generator::error retrieving entropy.`. 将 OpenSSL 更新到上游主机。
#8956 (阿列克谢-米洛维多夫)
- 执行时 `CREATE` 查询，在存储引擎参数中折叠常量表达式。将空数据库名称替换为当前数据库。修复 **#6508**,
#3492. 还修复了 `ClickHouseDictionarySource` 中检查本地地址。
#9262 (tabplibix)
- 修复段错误 `StorageMerge`，从 `StorageFile` 读取时可能发生。
#9387 (tabplibix)
- 防止丢失数据 `Kafka` 在极少数情况下，在读取后缀之后但在提交之前发生异常。修复 **#9378**. 相关: **#7175**
#9507 (菲利蒙诺夫)
- 修复尝试使用/删除时导致服务器终止的错误 `Kafka` 使用错误的参数创建的表。修复 **#9494**. 结合 **#9507**.
#9513 (菲利蒙诺夫)

新功能

- 添加 `deduplicate_blocks_in_dependent_materialized_views` 用于控制具有实例化视图的表中幂等插入的行为的选项。

这个新功能是由Altinity的特殊要求添加到错误修正版本中的。

#9070 (urykhy)

ClickHouse版本v20.1.2.4,2020-01-22

向后不兼容的更改

- 使设置 `merge_tree_uniform_read_distribution` 过时了 服务器仍可识别此设置，但无效。 #8308 (阿列克谢-米洛维多夫)
- 更改函数的返回类型 `greatCircleDistance` 到 `Float32` 因为现在计算的结果是 `Float32`. #7993 (阿列克谢-米洛维多夫)
- 现在预计查询参数表示为“escaped”格式。例如，要传递字符串 `a<tab>b` 你必须写 `a\tb` 或 `a\<tab>b` 并分别，`a%5Ctb` 或 `a%5C%09b` 在URL中。这是需要添加传递NULL作为的可能性 \N. 这修复 #7488. #8517 (阿列克谢-米洛维多夫)
- 启用 `use_minimalistic_part_header_in_zookeeper` 设置 `ReplicatedMergeTree` 默认情况下。这将显着减少存储在 ZooKeeper中的数据量。自19.1版本以来支持此设置，我们已经在多个服务的生产中使用它，半年以上没有任何问题。如果您有机会降级到19.1以前的版本，请禁用此设置。 #6850 (阿列克谢-米洛维多夫)
- 数据跳过索引已准备就绪并默认启用。设置 `allow_experimental_data_skipping_indices`, `allow_experimental_cross_to_join_conversion` 和 `allow_experimental_multiple_joins_emulation` 现在已经过时，什么也不做。 #7974 (阿列克谢-米洛维多夫)
- 添加新建 ANY JOIN 逻辑 `StorageJoin` 符合 JOIN 操作。要在不改变行为的情况下进行升级，您需要添加 SETTINGS `any_join_distinct_right_table_keys = 1` 引擎联接表元数据或在升级后重新创建这些表。 #8400 (Artem Zuikov)
- 要求重新启动服务器以应用日志记录配置中的更改。这是一种临时解决方法，可以避免服务器将日志记录到已删除的日志文件中的错误（请参阅 #8696). #8707 (Alexander Kuzmenkov)

新功能

- 添加了有关部件路径的信息 `system.merges`. #8043 (Vladimir Chebotarev)
- 添加执行能力 `SYSTEM RELOAD DICTIONARY` 查询中 `ON CLUSTER` 模式 #8288 (纪尧姆*塔瑟里)
- 添加执行能力 `CREATE DICTIONARY` 查询中 `ON CLUSTER` 模式 #8163 (阿利沙平)
- 现在用户的个人资料 `users.xml` 可以继承多个配置文件。 #8343 (Mikhail f. Shiryaev)
- 已添加 `system.stack_trace` 允许查看所有服务器线程的堆栈跟踪的表。这对于开发人员反省服务器状态非常有用。这修复 #7576. #8344 (阿列克谢-米洛维多夫)
- 添加 `DateTime64` 具有可配置子秒精度的数据类型。 #7170 (瓦西里*内姆科夫)
- 添加表函数 `clusterAllReplicas` 这允许查询集群中的所有节点。 #8493 (kiran sunkari)
- 添加聚合函数 `categoricalInformationValue` 其计算出离散特征的信息值。 #8117 (hcz)
- 加快数据文件的解析 `CSV`, `TSV` 和 `JSONEachRow` 通过并行进行格式化。 #7780 (Alexander Kuzmenkov)
- 添加功能 `bankerRound` 它执行银行家的四舍五入。 #8112 (hcz)
- 支持区域名称的嵌入式字典中的更多语言: 'ru', 'en', 'ua', 'uk', 'by', 'kz', 'tr', 'de', 'uz', 'lv', 'lt', 'et', 'pt', 'he', 'vi'. #8189 (阿列克谢-米洛维多夫)
- 改进的一致性 ANY JOIN 逻辑 现在 `t1 ANY LEFT JOIN t2` 等于 `t2 ANY RIGHT JOIN t1`. #7665 (Artem Zuikov)
- 添加设置 `any_join_distinct_right_table_keys` 这使旧的行为 ANY INNER JOIN. #7665 (Artem Zuikov)
- 添加新建 `SEMI` 和 `ANTI JOIN`. 老 ANY INNER JOIN 行为现在可作为 SEMI LEFT JOIN. #7665 (Artem Zuikov)
- 已添加 Distributed 格式 File 发动机和 file 表函数，它允许从读 .bin 通过异步插入生成的文件 Distributed 桌子 #8535 (尼古拉*科切托夫)
- 添加可选的重置列参数 `runningAccumulate` 这允许为每个新的键值重置聚合结果。 #8326 (谢尔盖*科诺年科)
- 添加使用ClickHouse作为普罗米修斯端点的能力。 #7900 (vdimir)
- 添加部分 `<remote_url_allow_hosts>` 在 `config.xml` 这将限制允许的主机用于远程表引擎和表函数 URL, S3, HDFS. #7154 (米哈伊尔*科罗托夫)
- 添加功能 `greatCircleAngle` 它计算球体上的距离（以度为单位）。 #8105 (阿列克谢-米洛维多夫)
- 改变地球半径与h3库一致。 #8105 (阿列克谢-米洛维多夫)
- 已添加 `JSONCompactEachRow` 和 `JSONCompactEachRowWithNamesAndTypes` 输入和输出格式。 #7841 (米哈伊尔*科罗托夫)
- 增加了与文件相关的表引擎和表函数的功能 (`File`, `S3`, `URL`, `HDFS`) 它允许读取和写入 `gzip` 基于附加引擎参数或文件扩展名的文件。 #7840 (安德烈*博德罗夫)
- 添加了 `randomASCII(length)` 函数，生成一个字符串与一个随机集 ASCII 可打印字符。 #8401 (刺刀)
- 添加功能 `JSONExtractArrayRaw` 它返回从未解析的json数组元素上的数组 JSON 字符串。 #8081 (Oleg Matrokhin)
- 添加 `arrayZip` 函数允许将多个长度相等的数组组合成一个元组数组。 #8149 (张冬)

- 添加根据配置的磁盘之间移动数据的能力 TTL-表达式为 *MergeTree 表引擎家族。#8140 (Vladimir Chebotarev)
- 增加了新的聚合功能 avgWeighted 其允许计算加权平均值。#7898 (安德烈·博德罗夫)
- 现在并行解析默认启用 TSV, TSKV, CSV 和 JSONEachRow 格式。#7894 (尼基塔·米哈伊洛夫)
- 从添加几个地理功能 H3 图书馆: h3GetResolution, h3EdgeAngle, h3EdgeLength, h3IsValid 和 h3kRing。#8034 (Konstantin Malanchev)
- 增加了对brotli的支持(br)压缩文件相关的存储和表函数。这修复 #8156. #8526 (阿列克谢-米洛维多夫)
- 添加 groupBit* 功能的 SimpleAggregationFunction 类型。#8485 (纪尧姆·塔瑟里)

错误修复

- 修复重命名表 Distributed 引擎 修复问题 #7868. #8306 (tavplubix)
- 现在字典支持 EXPRESSION 对于非ClickHouse SQL方言中任意字符串中的属性。#8098 (阿利沙平)
- 修复损坏 INSERT SELECT FROM mysql(...) 查询。这修复 #8070 和 #7960. #8234 (tavplubix)
- 修复错误 “Mismatch column sizes” 插入默认值时 Tuple 从 JSONEachRow. 这修复 #5653. #8606 (tavplubix)
- 现在将在使用的情况下抛出一个异常 WITH TIES 旁边的 LIMIT BY. 还增加了使用能力 TOP 与 LIMIT BY. 这修复 #7472. #7637 (尼基塔·米哈伊洛夫)
- 从新鲜的glibc版本中修复unintended依赖关系 clickhouse-odbc-bridge 二进制 #8046 (阿莫斯鸟)
- 修正错误的检查功能 *MergeTree 引擎家族. 现在, 当我们在最后一个颗粒和最后一个标记(非最终)中有相同数量的行时, 它不会失败。#8047 (阿利沙平)
- 修复插入 Enum* 列后 ALTER 查询, 当基础数值类型等于表指定类型时。这修复 #7836. #7908 (安东·波波夫)
- 允许非常数负 “size” 函数的参数 substring. 这是不允许的错误。这修复 #4832. #7703 (阿列克谢-米洛维多夫)
- 修复当错误数量的参数传递到解析错误 (OJ)DBC 表引擎。#7709 (阿利沙平)
- 将日志发送到syslog时使用正在运行的clickhouse进程的命令名。在以前的版本中, 使用空字符串而不是命令名称。#8460 (Michael Nacharov)
- 修复检查允许的主机 localhost. 这个公关修复了在提供的解决方案 #8241. #8342 (维塔利·巴拉诺夫)
- 修复罕见的崩溃 argMin 和 argMax 长字符串参数的函数, 当结果被用于 runningAccumulate 功能。这修复 #8325 #8341 (恐龙)
- 修复表的内存过度使用 Buffer 引擎 #8345 (Azat Khuzhin)
- 修正了可以采取的功能中的潜在错误 NULL 作为参数之一, 并返回非NULL。#8196 (阿列克谢-米洛维多夫)
- 在线程池中更好地计算后台进程的指标 MergeTree 表引擎。#8194 (Vladimir Chebotarev)
- 修复功能 IN 里面 WHERE 存在行级表筛选器时的语句。修复 #6687 #8357 (伊万)
- 现在, 如果整数值没有完全解析设置值, 则会引发异常。#7678 (米哈伊尔·科罗托夫)
- 修复当聚合函数用于查询具有两个以上本地分片的分布式表时出现的异常。#8164 (小路)
- 现在, bloom filter 可以处理零长度数组, 并且不执行冗余计算。#8242 (achimbab)
- 修正了通过匹配客户端主机来检查客户端主机是否允许 host_regexp 在指定 users.xml. #8241 (维塔利·巴拉诺夫)
- 放松不明确的列检查, 导致多个误报 JOIN ON 科。#8385 (Artem Zuikov)
- 修正了可能的服务器崩溃 (std::terminate) 当服务器不能发送或写入数据 JSON 或 XML 格式与值 String 数据类型 (需要 UTF-8 验证) 或使用Brotli算法或其他一些罕见情况下压缩结果数据时。这修复 #7603 #8384 (阿列克谢-米洛维多夫)
- 修复竞争条件 StorageDistributedDirectoryMonitor 被线人发现 这修复 #8364. #8383 (尼古拉·科切托夫)
- 现在背景合并 *MergeTree 表引擎家族更准确地保留存储策略卷顺序。#8549 (Vladimir Chebotarev)
- 现在表引擎 Kafka 与正常工作 Native 格式。这修复 #6731 #7337 #8003. #8016 (filimonov)
- 固定格式与标题 (如 CSVWithNames) 这是抛出关于EOF表引擎的异常 Kafka. #8016 (filimonov)
- 修复了从子查询右侧部分制作set的错误 IN 科。这修复 #5767 和 #2542. #7755 (尼基塔·米哈伊洛夫)
- 从存储读取时修复可能的崩溃 File. #7756 (尼古拉·科切托夫)
- 在固定的文件读取 Parquet 包含类型列的格式 list. #8334 (马苏兰)
- 修复错误 Not found column 对于分布式查询 PREWHERE 条件取决于采样键 if max_parallel_replicas > 1. #7913 (尼古拉·科切托夫)
- 修复错误 Not found column 如果使用查询 PREWHERE 依赖于表的别名, 结果集由于主键条件而为空。#7911 (尼古拉·科切托夫)
- 函数的固定返回类型 rand 和 randConstant 在情况下 Nullable 争论。现在函数总是返回 UInt32 而且从来没有 Nullable(UInt32). #8204 (尼古拉·科切托夫)
- 禁用谓词下推 WITH FILL 表达。这修复 #7784. #7789 (张冬)
- 修正错误 count() 结果 SummingMergeTree 当 FINAL 部分被使用。#3280 #7786 (尼基塔·米哈伊洛夫)
- 修复来自远程服务器的常量函数可能不正确的结果。它发生在具有以下功能的查询中 version(), uptime() 等。它为不同的服务提供不同的常量值。修复旨 #7600 #7600 (尼基塔·米哈伊洛夫)

向的服分命达四个向的市里但。达修及 #1000, #1001 (七叶树*竹叶大)

- 修复下推谓词优化中导致错误结果的复杂错误。这解决了下推谓词优化的很多问题。#8503 (张冬)
- 修复崩溃 CREATE TABLE .. AS dictionary 查询。#8508 (Azat Khuzhin)
- 一些改进ClickHouse语法 .g4 文件 #8294 (太阳里)
- 修复导致崩溃的错误 JOIN 与表与发动机 Join. 这修复 #7556 #8254 #7915 #8100. #8298 (Artem Zuikov)
- 修复冗余字典重新加载 CREATE DATABASE. #7916 (Azat Khuzhin)
- 限制从读取流的最大数量 StorageFile 和 StorageHDFS. 修复
<https://github.com/ClickHouse/ClickHouse/issues/7650>. #7981 (阿利沙平)
- 修复bug ALTER ... MODIFY ... CODEC 查询，当用户同时指定默认表达式和编解ec。修复 8593. #8614 (阿利沙平)
- 修复列的后台合并错误 SimpleAggregateFunction(LowCardinality) 类型。#8613 (尼古拉*科切托夫)
- 固定类型签入功能 toDateTime64. #8375 (瓦西里*内姆科夫)
- 现在服务器不崩溃 LEFT 或 FULL JOIN 与和加入引擎和不支持 join_use_nulls 设置。#8479 (Artem Zuikov)
- 现在 DROP DICTIONARY IF EXISTS db.dict 查询不会抛出异常，如果 db 根本不存在 #8185 (维塔利*巴拉诺夫)
- 修复表函数中可能出现的崩溃 (file, mysql, remote) 引用删除引起的 IStorage 对象。修复插入表函数时指定的列的不正确解析。#7762 (tavplubix)
- 确保网络启动前 clickhouse-server. 这修复 #7507. #8570 (余志昌)
- 修复安全连接的超时处理，因此查询不会无限挂起。这修复 #8126. #8128 (阿列克谢-米洛维多夫)
- 修复 clickhouse-copier 并发工人之间的冗余争用。#7816 (丁香飞)
- 现在突变不会跳过附加的部分，即使它们的突变版本比当前的突变版本大。#7812 (余志昌) #8250 (阿利沙平)
- 忽略冗余副本 *MergeTree 数据部分移动到另一个磁盘和服务器重新启动后。#7810 (Vladimir Chebotarev)
- 修复崩溃 FULL JOIN 与 LowCardinality 在 JOIN 钥匙 #8252 (Artem Zuikov)
- 禁止在插入查询中多次使用列名，如 INSERT INTO tbl (x, y, x). 这修复 #5465, #7681. #7685 (阿利沙平)
- 增加了回退，用于检测未知Cpu的物理CPU内核数量（使用逻辑CPU内核数量）。这修复 #5239. #7726 (阿列克谢-米洛维多夫)
- 修复 There's no column 实例化列和别名列错。#8210 (Artem Zuikov)
- 固定切断崩溃时 EXISTS 查询没有使用 TABLE 或 DICTIONARY 预选赛就像 EXISTS t. 这修复 #8172. 此错误在版本 19.17 中引入。#8213 (阿列克谢-米洛维多夫)
- 修复罕见错误 "Sizes of columns doesn't match" 使用时可能会出现 SimpleAggregateFunction 列。#7790 (Boris Granveaud)
- 修正错误，其中用户空 allow_databases 可以访问所有数据库（和相同的 allow_dictionaries）。#7793 (DeifyTheGod)
- 修复客户端崩溃时，服务器已经从客户端断开连接。#8071 (Azat Khuzhin)
- 修复 ORDER BY 在按主键前缀和非主键后缀排序的情况下行为。#7759 (安东*波波夫)
- 检查表中是否存在合格列。这修复 #6836. #7758 (Artem Zuikov)
- 固定行为 ALTER MOVE 合并完成后立即运行移动指定的超部分。修复 #8103. #8104 (Vladimir Chebotarev)
- 使用时修复可能的服务器崩溃 UNION 具有不同数量的列。修复 #7279. #7929 (尼古拉*科切托夫)
- 修复函数结果子字符串的大小 substr 负大小。#8589 (尼古拉*科切托夫)
- 现在服务器不执行部分突变 MergeTree 如果后台池中没有足够的可用线程。#8588 (tavplubix)
- 修复格式化时的小错字 UNION ALL AST. #7999 (lita091)
- 修正了负数不正确的布隆过滤结果。这修复 #8317. #8566 (张冬)
- 在解压缩固定潜在的缓冲区溢出。恶意用户可以传递捏造的压缩数据，这将导致缓冲区后读取。这个问题是由Yandex 信息安全部队的Eldar Zaitov发现的。#8404 (阿列克谢-米洛维多夫)
- 修复因整数溢出而导致的错误结果 arrayIntersect. #7777 (尼古拉*科切托夫)
- 现在 OPTIMIZE TABLE query 不会等待脱机副本执行该操作。#8314 (javi santana)
- 固定 ALTER TTL 解析器 Replicated*MergeTree 桌子 #8318 (Vladimir Chebotarev)
- 修复服务器和客户端之间的通信，以便服务器在查询失败后读取临时表信息。#8084 (Azat Khuzhin)
- 修复 bitmapAnd 在聚合位图和标量位图相交时出现函数错误。#8082 (黄月)
- 完善的定义 ZXid 根据动物园管理员的程序员指南，它修复了错误 clickhouse-cluster-copier. #8088 (丁香飞)
- odbc 表函数现在尊重 external_table_functions_use_nulls 设置。#7506 (瓦西里*内姆科夫)
- 修正了导致罕见的数据竞赛的错误。#8143 (亚历山大*卡扎科夫)
- 现在 SYSTEM RELOAD DICTIONARY 完全重新加载字典，忽略 update_field. 这修复 #7440. #8037 (维塔利*巴拉诺夫)
- 添加检查字典是否存在于创建查询的能力。#8032 (阿利沙平)
- 修复 Float* 解析中 Values 格式。这修复 #7817. #7870 (tavplubix)
- 修复崩溃时，我们不能在一些后台操作保留空间 *MergeTree 表引擎家族。#7873 (Vladimir Chebotarev)
- 修复表包含合并操作时的崩溃 SimpleAggregateFunction(LowCardinality) 列。这修复 #8515. #8522 (Azat

Khuzhin)

- 恢复对所有ICU区域设置的支持，并添加对常量表达式应用排序规则的功能。还添加语言名称 `system.collations` 桌子 #8051 (阿利沙平)
- 修正错误时，外部字典与零最小寿命 (`LIFETIME(MIN 0 MAX N)`, `LIFETIME(N)`) 不要在后台更新。#7983 (阿利沙平)
- 修复当clickhouse源外部字典在查询中有子查询时崩溃。#8351 (尼古拉*科切托夫)
- 修复文件扩展名不正确的解析表与引擎 URL. 这修复 #8157. #8419 (安德烈*博德罗夫)
- 修复 `CHECK TABLE` 查询为 *MergeTree 表没有关键. 修复 #7543. #7979 (阿利沙平)
- 固定转换 `Float64` 到MySQL类型。#8079 (尤里*巴拉诺夫)
- 现在，如果表没有完全删除，因为服务器崩溃，服务器将尝试恢复并加载它。#8176 (tavplubix)
- 修复了表函数中的崩溃 file 同时插入到不存在的文件。现在在这种情况下，文件将被创建，然后插入将被处理。#8177 (Olga Khvostikova)
- 修复罕见的死锁时，可能发生 `trace_log` 处于启用状态。#7838 (filimonov)
- 添加能力与不同类型的工作，除了 `Date` 在 `RangeHashed` 从DDL查询创建的外部字典。修复 7899. #8275 (阿利沙平)
- 修复崩溃时 `now64()` 用另一个函数的结果调用。#8270 (瓦西里*内姆科夫)
- 修正了通过mysql有线协议检测客户端IP连接的错误。#7743 (Dmitry Muzyka)
- 修复空阵列处理 `arraySplit` 功能。这修复 #7708. #7747 (hcZ)
- 修复了以下问题 `pid-file` 另一个运行 `clickhouse-server` 可能会被删除。#8487 (徐伟清)
- 修复字典重新加载，如果它有 `invalidate_query`，停止更新，并在以前的更新尝试一些异常。#8029 (阿利沙平)
- 修正了功能错误 `arrayReduce` 这可能会导致“double free”和聚合函数组合器中的错误 `Resample` 这可能会导致内存泄漏。添加聚合功能 `aggThrow`. 此功能可用于测试目的。#8446 (阿列克谢-米洛维多夫)

改进

- 改进了使用时的日志记录 `S3` 表引擎。#8251 (Grigory Pervakov)
- 在调用时未传递任何参数时打印帮助消息 `clickhouse-local`. 这修复 #5335. #8230 (安德烈*纳戈尔尼)
- 添加设置 `mutations_sync` 这允许等待 `ALTER UPDATE/DELETE` 同步查询。#8237 (阿利沙平)
- 允许设置相对 `user_files_path` 在 `config.xml` (在类似的方式 `format_schema_path`). #7632 (hcZ)
- 为转换函数添加非法类型的异常 -OrZero 后缀 #7880 (安德烈*科尼亞耶夫)
- 简化在分布式查询中发送到分片的数据头的格式。#8044 (维塔利*巴拉诺夫)
- `Live View` 表引擎重构。#8519 (vzakaznikov)
- 为从DDL查询创建的外部字典添加额外的检查。#8127 (阿利沙平)
- 修复错误 `Column ... already exists` 使用时 `FINAL` 和 `SAMPLE` together, e.g. `select count() from table final sample 1/2.` 修复 #5186. #7907 (尼古拉*科切托夫)
- 现在表的第一个参数 `joinGet` 函数可以是表标识符。#7707 (阿莫斯鸟)
- 允许使用 `MaterializedView` 与上面的子查询 `Kafka` 桌子 #8197 (filimonov)
- 现在后台在磁盘之间移动，运行它的seprate线程池。#7670 (Vladimir Chebotarev)
- `SYSTEM RELOAD DICTIONARY` 现在同步执行。#8240 (维塔利*巴拉诺夫)
- 堆栈跟踪现在显示物理地址 (对象文件中的偏移量)，而不是虚拟内存地址 (加载对象文件的位置)。这允许使用 `addr2line` 当二进制独立于位置并且ASLR处于活动状态时。这修复 #8360. #8387 (阿列克谢-米洛维多夫)
- 支持行级安全筛选器的新语法: `<table name='table_name'>...</table>`. 修复 #5779. #8381 (伊万)
- 现在 `cityHash` 功能可以与工作 `Decimal` 和 `UUID` 类型。修复 #5184. #7693 (米哈伊尔*科罗托夫)
- 从系统日志中删除了固定的索引粒度 (它是1024)，因为它在实现自适应粒度之后已经过时。#7698 (阿列克谢-米洛维多夫)
- 当ClickHouse在没有SSL的情况下编译时，启用MySQL兼容服务器。#7852 (尤里*巴拉诺夫)
- 现在服务器校验和分布式批处理，这在批处理中损坏数据的情况下提供了更多详细的错误。#7914 (Azat Khuzhin)
- 碌莽祿Support: `DROP DATABASE`, `DETACH TABLE`, `DROP TABLE` 和 `ATTACH TABLE` 为 MySQL 数据库引擎。#8202 (张冬)
- 在S3表功能和表引擎中添加身份验证。#7623 (Vladimir Chebotarev)
- 增加了检查额外的部分 `MergeTree` 在不同的磁盘上，为了不允许错过未定义磁盘上的数据部分。#8118 (Vladimir Chebotarev)
- 启用Mac客户端和服务器的SSL支持。#8297 (伊万)
- 现在ClickHouse可以作为MySQL联合服务器 (参见<https://dev.mysql.com/doc/refman/5.7/en/federated-create-server.html>)。#7717 (Maxim Fedotov)
- `clickhouse-client` 现在只能启用 `bracketed-paste` 当多查询处于打开状态且多行处于关闭状态时。这修复 (#7757)
<https://github.com/ClickHouse/ClickHouse/pull/7757>。#7761 (阿吉斯拉夫)

<https://github.com/ClickHouse/ClickHouse/issues/1151> #1151 (门关列与)

- 确保 Support: Array(Decimal) 在 if 功能。 #7721 (Artem Zuikov)
- 支持小数 arrayDifference, arrayCumSum 和 arrayCumSumNegative 功能。 #7724 (Artem Zuikov)
- 已添加 lifetime 列到 system.dictionaries 桌子 #6820 #7727 (kekekekule)
- 改进了检查不同磁盘上的现有部件 *MergeTree 表引擎. 地址 #7660. #8440 (Vladimir Chebotarev)
- 集成与 AWS SDK 为 S3 交互允许使用开箱即用的所有S3功能。 #8011 (帕维尔*科瓦连科)
- 增加了对子查询的支持 Live View 桌子 #7792 (vzakaznikov)
- 检查使用 Date 或 DateTime 从列 TTL 表达式已删除。 #7920 (Vladimir Chebotarev)
- 有关磁盘的信息已添加到 system.detached_parts 桌子 #7833 (Vladimir Chebotarev)
- 现在设置 max_(table|partition)_size_to_drop 无需重新启动即可更改。 #7779 (Grigory Pervakov)
- 错误消息的可用性略好。 要求用户不要删除下面的行 Stack trace:.. #7897 (阿列克谢-米洛维多夫)
- 更好地阅读消息 Kafka 引擎在各种格式后 #7935. #8035 (伊万)
- 与不支持MySQL客户端更好的兼容性 sha2_password 验证插件。 #8036 (尤里*巴拉诺夫)
- 支持MySQL兼容性服务器中的更多列类型。 #7975 (尤里*巴拉诺夫)
- 执行 ORDER BY 优化 Merge, Buffer 和 Materialized View 存储与底层 MergeTree 桌子 #8130 (安东*波波夫)
- 现在我们总是使用POSIX实现 getrandom 与旧内核更好的兼容性 (\<3.17)。 #7940 (阿莫斯鸟)
- 更好地检查移动ttl规则中的有效目标。 #8410 (Vladimir Chebotarev)
- 更好地检查损坏的刀片批次 Distributed 表引擎。 #7933 (Azat Khuzhin)
- 添加带有部件名称数组的列，这些部件将来必须处理突变 system.mutations 桌子 #8179 (阿利沙平)
- 处理器的并行合并排序优化。 #8552 (尼古拉*科切托夫)
- 设置 mark_cache_min_lifetime 现在已经过时了，什么也不做。 在以前的版本中，标记缓存可以在内存中增长大于 mark_cache_size 以容纳内的数据 mark_cache_min_lifetime 秒。 这导致了混乱和比预期更高的内存使用率，这在内存受限的系统上尤其糟糕。 如果您在安装此版本后会看到性能下降，则应增加 mark_cache_size. #8484 (阿列克谢-米洛维多夫)
- 准备使用 tid 到处都是 这是必要的 #7477. #8276 (阿列克谢-米洛维多夫)

性能改进

- 处理器管道中的性能优化。 #7988 (尼古拉*科切托夫)
- 缓存字典中过期密钥的非阻塞更新（具有读取旧密钥的权限）。 #8303 (尼基塔*米哈伊洛夫)
- 没有编译ClickHouse -fno-omit-frame-pointer 在全球范围内多余一个寄存器。 #8097 (阿莫斯鸟)
- 加速 greatCircleDistance 功能，并为它添加性能测试。 #7307 (Olga Khvostikova)
- 改进的功能性能 roundDown. #8465 (阿列克谢-米洛维多夫)
- 改进的性能 max, min, argMin, argMax 为 DateTime64 数据类型。 #8199 (瓦西里*内姆科夫)
- 改进了无限制或大限制和外部排序的排序性能。 #8545 (阿列克谢-米洛维多夫)
- 改进的性能格式化浮点数高达6倍。 #8542 (阿列克谢-米洛维多夫)
- 改进的性能 modulo 功能。 #7750 (阿莫斯鸟)
- 优化 ORDER BY 并与单列键合并。 #8335 (阿列克谢-米洛维多夫)
- 更好地实施 arrayReduce, -Array 和 -State 组合子 #7710 (阿莫斯鸟)
- 现在 PREWHERE 应优化为至少一样高效 WHERE. #7769 (阿莫斯鸟)
- 改进方式 round 和 roundBankers 处理负数。 #8229 (hcz)
- 改进的解码性能 DoubleDelta 和 Gorilla 编解码器大约30-40%。 这修复 #7082. #8019 (瓦西里*内姆科夫)
- 改进的性能 base64 相关功能。 #8444 (阿列克谢-米洛维多夫)
- 增加了一个功能 geoDistance. 它类似于 greatCircleDistance 但使用近似于WGS-84椭球模型。 两个功能的性能几乎相同。 #8086 (阿列克谢-米洛维多夫)
- 更快 min 和 max 聚合函数 Decimal 数据类型。 #8144 (Artem Zuikov)
- 矢量化处理 arrayReduce. #7608 (阿莫斯鸟)
- if 链现在优化为 multilf. #8355 (kamalov-ruslan)
- 修复性能回归 Kafka 表引擎在19.15中引入。 这修复 #7261. #7935 (filimonov)
- 已删除 “pie” 代码生成 gcc 从Debian软件包偶尔带来默认情况下。 #8483 (阿列克谢-米洛维多夫)
- 并行解析数据格式 #6553 (尼基塔*米哈伊洛夫)
- 启用优化的解析器 Values 默认使用表达式 (input_format_values_deduce_templates_of_expressions=1). #8231 (tavplubix)

构建/测试/包装改进

- 构建修复 ARM 而在最小模式。 #8304 (proller)

- 添加复盖文件刷新 `clickhouse-server` 当不调用 `std::atexit` 时。还略微改进了无状态测试的复盖率日志记录。 #8267 (阿利沙平)
- 更新 `contrib` 中的 LLVM 库。避免从操作系统包中使用 LLVM。#8258 (阿列克谢-米洛维多夫)
- 使 `bund` 绑 `curl` 建立完全安静。#8232 #8203 (帕维尔*科瓦连科)
- 修复一些 `MemorySanitizer` 警告。#8235 (Alexander Kuzmenkov)
- 使用 `add_warning` 和 `no_warning` 宏 `CMakeLists.txt`. #8604 (伊万)
- 添加对 Minio S3 兼容对象的支持 (<https://min.io/>) 为了更好的集成测试。#7863 #7875 (帕维尔*科瓦连科)
- 导入 `libc` 标题到 `contrib`。它允许在各种系统中使构建更加一致 (仅适用于 `x86_64-linux-gnu`). #5773 (阿列克谢-米洛维多夫)
- 删除 `-fPIC` 从一些图书馆。#8464 (阿列克谢-米洛维多夫)
- 清洁 `CMakeLists.txt` 对于卷曲。看<https://github.com/ClickHouse/ClickHouse/pull/8011#issuecomment-569478910> #8459 (阿列克谢-米洛维多夫)
- 无声警告 `CapNProto` 图书馆。#8220 (阿列克谢-米洛维多夫)
- 为短字符串优化哈希表添加性能测试。#7679 (阿莫斯鸟)
- 现在 ClickHouse 将建立在 AArch64 即使 `MADV_FREE` 不可用。这修复 #8027. #8243 (阿莫斯鸟)
- 更新 `zlib-ng` 来解决记忆消毒的问题 #7182 #8206 (Alexander Kuzmenkov)
- 在非 Linux 系统上启用内部 MySQL 库，因为操作系统包的使用非常脆弱，通常根本不起作用。这修复 #5765. #8426 (阿列克谢-米洛维多夫)
- 修复了启用后在某些系统上构建的问题 `libc++`. 这取代了 #8374. #8380 (阿列克谢-米洛维多夫)
- 眉眉露>> `Field` 方法更类型安全，以找到更多的错误。#7386 #8209 (Alexander Kuzmenkov)
- 添加丢失的文件到 `libc-headers` 子模块。#8507 (阿列克谢-米洛维多夫)
- 修复错误 `JSON` 引用性能测试输出。#8497 (尼古拉*科切托夫)
- 现在堆栈跟踪显示 `std::exception` 和 `Poco::Exception`. 在以前的版本中，它仅适用于 `DB::Exception`. 这改进了诊断。#8501 (阿列克谢-米洛维多夫)
- 移植 `clock_gettime` 和 `clock_nanosleep` 对于新鲜的 glibc 版本。#8054 (阿莫斯鸟)
- 启用 `part_log` 在示例配置开发人员。#8609 (阿列克谢-米洛维多夫)
- 修复重新加载的异步性质 `01036_no_superfluous_dict_reload_on_create_database*`. #8111 (Azat Khuzhin)
- 固定编解码器性能测试。#8615 (瓦西里*内姆科夫)
- 添加安装脚本 `.tgz` 为他们构建和文档。#8612 #8591 (阿利沙平)
- 删除旧 ZSTD 测试 (它是在 2016 年创建的，以重现 zstd1.0 版本之前的错误)。这修复 #8618. #8619 (阿列克谢-米洛维多夫)
- 固定构建在 Mac OS 卡特琳娜。#8600 (meo)
- 增加编解码器性能测试中的行数，以便结果显着。#8574 (瓦西里*内姆科夫)
- 在调试版本中，处理 `LOGICAL_ERROR` 异常作为断言失败，使得它们更容易被注意到。#8475 (Alexander Kuzmenkov)
- 使与格式相关的性能测试更具确定性。#8477 (阿列克谢-米洛维多夫)
- 更新 `lz4` 来修复记忆消毒器的故障 #8181 (Alexander Kuzmenkov)
- 在异常处理中抑制已知 `MemorySanitizer` 误报。#8182 (Alexander Kuzmenkov)
- 更新 `gcc` 和 `g++` 到版本 9 在 `build/docker/build.sh` #7766 (TLightSky)
- 添加性能测试用例来测试 `PREWHERE` 比 `WHERE`. #7768 (阿莫斯鸟)
- 在修复一个笨拙的测试方面取得了进展。#8621 (阿列克谢-米洛维多夫)
- 避免从 `MemorySanitizer` 报告数据 `libunwind`. #8539 (阿列克谢-米洛维多夫)
- 更新 `libc++` 到最新版本。#8324 (阿列克谢-米洛维多夫)
- 从源头构建 ICU 库。这修复 #6460. #8219 (阿列克谢-米洛维多夫)
- 从切换 `libressl` 到 `openssl`. ClickHouse 应在此更改后支持 TLS1.3 和 SNI。这修复 #8171. #8218 (阿列克谢-米洛维多夫)
- 使用时固定的 UBSan 报告 `chacha20_poly1305` 从 SSL (发生在连接到 <https://yandex.ru/>)。#8214 (阿列克谢-米洛维多夫)
- 修复默认密码文件的模式 `.deb` linux 发行版。#8075 (proller)
- 改进的表达式获取 `clickhouse-server` PID 输入 `clickhouse-test`. #8063 (亚历山大*卡扎科夫)
- 更新 `contrib/googletest` 到 v1.10.0。#8587 (Alexander Burmak)
- 修复了 ThreadSanitizer 报告 `base64` 图书馆。还将此库更新到最新版本，但无关紧要。这修复 #8397. #8403 (阿列克谢-米洛维多夫)
- 修复 `00600_replace_running_query` 对于处理器。#8272 (尼古拉*科切托夫)

- 刚刚又付 `tcmalloc` 从 `ICUmakeLISTS.TXT` 变回干 `#0010` (门到无刚-不谷维夕大) #8311 (阿列克谢-米洛维多夫)
- 发布海湾合作委员会构建现在使用 `libc++` 而不是 `libstdc++`. 最近 `libc++` 只与叮当一起使用。这将提高构建配置的一致性和可移植性。#8222 (阿列克谢-米洛维多夫)
- 使用MemorySanitizer启用ICU库进行构建。#8224 (阿列克谢-米洛维多夫)
- 删除代码的特殊情况 `tcmalloc`, 因为它不再受支持。#8225 (阿列克谢-米洛维多夫)
- 在CI coverage任务中, 优雅地终止服务器以允许它保存coverage报告。这修复了我们最近看到的不完整的覆盖率报告。#8142 (阿利沙平)
- 针对所有编解码器的性能测试 `Float64` 和 `UInt64` 值。#8349 (瓦西里*内姆科夫)
- `termcap` 非常不推荐使用, 并导致各种问题 (f.g.missing “up” 帽和呼应 `\J` 而不是多行)。帮个忙 `terminfo` 或 `bund` 绑 `ncurses`. #7737 (阿莫斯鸟)
- 修复 `test_storage_s3` 集成测试。#7734 (尼古拉*科切托夫)
- 碌莽禄Support: `StorageFile(<format>, null)` 将块插入给定格式的文件而不实际写入磁盘。这是性能测试所必需的。#8455 (阿莫斯鸟)
- 添加参数 `--print-time` 功能测试打印每个测试的执行时间。#8001 (尼古拉*科切托夫)
- 添加断言 `KeyCondition` 同时评估RPN。这将修复来自gcc-9的警告。#8279 (阿列克谢-米洛维多夫)
- 在CI构建中转储cmake选项。#8273 (Alexander Kuzmenkov)
- 不要为某些fat库生成调试信息。#8271 (阿列克谢-米洛维多夫)
- 赖眉露>> `log_to_console.xml` 始终登录到stderr, 无论它是否交互。#8395 (Alexander Kuzmenkov)
- 删除了一些未使用的功能 `clickhouse-performance-test` 工具 #8555 (阿列克谢-米洛维多夫)
- 现在我们也将搜索 `lld-X` 与相应的 `clang-X` 版本。#8092 (阿利沙平)
- 实木复合地板建设改善。#8421 (马苏兰)
- 更多海湾合作委员会警告 #8221 (kreuzerkrieg)
- Arch Linux的软件包现在允许运行ClickHouse服务器, 而不仅仅是客户端。#8534 (Vladimir Chebotarev)
- 修复与处理器的测试。微小的性能修复。#7672 (尼古拉*科切托夫)
- 更新contrib/protobuf。#8256 (Matwey V.Kornilov)
- 在准备切换到c++20作为新年庆祝活动。“May the C++ force be with ClickHouse.” #8447 (阿莫斯鸟)

实验特点

- 增加了实验设置 `min_bytes_to_use_mmap_io`. 它允许读取大文件, 而无需将数据从内核复制到用户空间。默认情况下禁用该设置。建议的阈值大约是64MB, 因为mmap/munmap很慢。#8520 (阿列克谢-米洛维多夫)
- 返工配额作为访问控制系统的一部分。增加了新表 `system.quotas`, 新功能 `currentQuota`, `currentQuotaKey`, 新的SQL语法 `CREATE QUOTA`, `ALTER QUOTA`, `DROP QUOTA`, `SHOW QUOTA`. #7257 (维塔利*巴拉诺夫)
- 允许跳过带有警告的未知设置, 而不是引发异常。#7653 (维塔利*巴拉诺夫)
- 重新设计的行策略作为访问控制系统的一部分。增加了新表 `system.row_policies`, 新功能 `currentRowPolicies()`, 新的SQL语法 `CREATE POLICY`, `ALTER POLICY`, `DROP POLICY`, `SHOW CREATE POLICY`, `SHOW POLICIES`. #7808 (维塔利*巴拉诺夫)

安全修复

- 修正了读取目录结构中的表的可能性 `File` 表引擎。这修复 #8536. #8537 (阿列克谢-米洛维多夫)

更新日志2019

MySQL接口

ClickHouse支持MySQL线协议。它可以通过启用 `mysql_port` 在配置文件中设置:

```
<mysql_port>9004</mysql_port>
```

使用命令行工具连接的示例 `mysql`:

```
$ mysql --protocol tcp -u default -P 9004
```

如果连接成功，则输出：

```
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 4  
Server version: 20.2.1.1-ClickHouse  
  
Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

为了与所有MySQL客户端兼容，建议使用以下命令指定用户密码 双SHA1 在配置文件中。
如果使用用户密码指定 SHA256，一些客户端将无法进行身份验证（mysqljs和旧版本的命令行工具mysql）。

限制：

- 不支持准备好的查询
- 某些数据类型以字符串形式发送

来自第三方开发人员的代理服务器

chproxy 是ClickHouse数据库的http代理和负载均衡器。

特征

每用户路由和响应缓存。

灵活的限制。

*自动SSL证书续订。

在Go中实现。

KittenHouse

KittenHouse 设计为ClickHouse和应用程序服务器之间的本地代理，以防在应用程序端缓冲INSERT数据是不可能或不方便的。

特征：

内存和磁盘数据缓冲。

每表路由。

*负载平衡和健康检查。

在Go中实现。

ツ环板-ヨツ嘉ツツ偲

ツ环板-ヨツ嘉ツツ偲 是一个简单的ClickHouse插入收集器。

特征：

分组请求并按阈值或间隔发送。

多个远程服务器。

*基本身份验证。

在Go中实现。

第三方开发的可视化界面

开源

Tabix

ClickHouse Web 界面 [Tabix](#).

主要功能：

- 浏览器直接连接 ClickHouse，不需要安装其他软件。
- 高亮语法的编辑器。
- 自动命令补全。
- 查询命令执行的图形分析工具。
- 配色方案选项。

[Tabix 文档](#).

HouseOps

[HouseOps](#) 是一个交互式 UI/IDE 工具，可以运行在 OSX, Linux and Windows 平台中。

主要功能：

- 查询高亮语法提示，可以以表格或 JSON 格式查看数据。
- 支持导出 CSV 或 JSON 格式数据。
- 支持查看查询执行的详情，支持 KILL 查询。
- 图形化显示，支持显示数据库中所有的表和列的详细信息。
- 快速查看列占用的空间。
- 服务配置。

以下功能正在计划开发：

- 数据库管理
- 用户管理
- 实时数据分析
- 集群监控
- 集群管理
- 监控副本情况以及 Kafka 引擎表

灯塔

[灯塔](#) 是ClickHouse的轻量级Web界面。

特征：

- 包含过滤和元数据的表列表。
- 带有过滤和排序的表格预览。
- 只读查询执行。

DBeaver

[DBeaver](#) 具有ClickHouse支持的通用桌面数据库客户端。

特征：

- 使用语法高亮显示查询开发。
- 表格预览。
- 自动完成。

ツ环板-ヨツ嘉ツツ偲

ツ环板-ヨツ嘉ツツ偲 是ClickHouse的替代命令行客户端，用Python 3编写。

特征：

- 自动完成。
- 查询和数据输出的语法高亮显示。
- 寻呼机支持数据输出。
- 自定义PostgreSQL类命令。

ツ暗エツ氾环催ツ団ツ法ツ人

[clickhouse-flamegraph](<https://github.com/Slach/clickhouse-flamegraph>) 是一个可视化的专业工具`system.trace_log`如[flamegraph](<http://www.brendangregg.com/flamegraphs.html>).

商业

ツ环板Softwareヨツ嘉ツ

整体学 在2019年被Gartner FrontRunners列为可用性最高排名第二的商业智能工具之一。 Holistics是一个基于SQL的全栈数据平台和商业智能工具，用于设置您的分析流程。

特征：

- 自动化的电子邮件，Slack和Google表格报告时间表。
- 强大的SQL编辑器，具有版本控制，自动完成，可重用的查询组件和动态过滤器。
- 通过iframe在自己的网站或页面中嵌入仪表板。
- 数据准备和ETL功能。
- SQL数据建模支持数据的关系映射。

DataGrip

DataGrip 是JetBrains的数据库IDE，专门支持ClickHouse。 它还嵌入到其他基于IntelliJ的工具中：PyCharm，IntelliJ IDEA，GoLand，PhpStorm等。

特征：

- 非常快速的代码完成。
- ClickHouse语法高亮显示。
- 支持ClickHouse特有的功能，例如嵌套列，表引擎。
- 数据编辑器。
- 重构。
- 搜索和导航。

第三方开发的库

放弃

Yandex不维护下面列出的库，也没有进行任何广泛的测试以确保其质量。

- Python
 - [infi.clickhouse_orm](#)
 - [ツ环板driverヨツ嘉ツツ偲](#)
 - [ツ环板clientヨツ嘉ツツ偲](#)
- PHP
 - [smi2/phpclickhouse](#)

- [8bitov/clickhouse-php](#) 客户端
- [ツ暗エツ汎环催ツ団ツ法ツ人](#)
- [ツ环板clientヨツ嘉ツツ偲](#)
- [seva-code/php-click-house-client](#)
- [ツ环板clientヨツ嘉ツツ偲](#)
- 走吧
 - [clickhouse](#)
 - [ツ环板-ヨツ嘉ツツ偲](#)
 - [ツ暗エツ汎环催ツ団ツ法ツ人](#)
 - [golang-clickhouse](#)
- NodeJs
 - [ツ暗エツ汎环催ツ団ツ法ツ人](#))
 - [ツ环板-ヨツ嘉ツツ偲](#)
- Perl
 - [perl-DBD-ClickHouse](#)
 - [HTTP-ClickHouse](#)
 - [ツ暗エツ汎环催ツ団ツ法ツ人](#)
- Ruby
 - [ツ暗エツ汎环催ツ団\)](#)
 - [ツ暗エツ汎环催ツ団ツ法ツ人](#)
- R
 - [clickhouse-r](#)
 - [RClickhouse](#)
- Java
 - [clickhouse-client-java](#)
- 斯卡拉
 - [掳胫client-禄脢鹿脷露胫鲁隆鹿-client酶](#)
- Kotlin
 - [AORM](#)
- C#
 - [克莱克豪斯Ado](#)
 - [ClickHouse.Net](#)
 - [克莱克豪斯客户](#)
- 仙丹
 - [clickhousex](#)
- 尼姆
 - [nim-clickhouse](#)

第三方集成库

声明

Yandex不维护下面列出的库，也没有进行任何广泛的测试以确保其质量。

基建产品

- 关系数据库管理系统
 - [MySQL](#)
 - [ProxySQL](#)
 - [clickhouse-mysql-data-reader](#)
 - [horgh-复制器](#)
 - [PostgreSQL](#)
 - [clickhousedb_fdw](#)
 - [infi.clickhouse_fdw](#) (使用 [infi.clickhouse_orm](#))
 - [noco](#)

- [pyclick](#)
- [MSSQL](#)
 - [ClickHouseMigrator](#)
- 消息队列
 - [卡夫卡](#)
 - [clickhouse_sinker](#) (使用 [去客户](#))
 - [stream-loader-clickhouse](#)
- 流处理
 - [Flink](#)
 - [flink-clickhouse-sink](#)
- 对象存储
 - [S3](#)
 - [对象存储backup](#)
- 容器编排
 - [Kubernetes](#)
 - [clickhouse-操作](#)
- 配置管理
 - [木偶](#)
 - [配置中心](#)
 - [mfedorov\(clickhouse\)](#)
- 监控
 - [石墨](#)
 - [graphouse](#)
 - [Graphite](#)
 - [GraphiteMergeTree](#)
 - [优化静态分区](#)
 - [如果从规则汇总配置可以应用](#)
 - [Grafana](#)
 - [clickhouse-grafana](#)
 - [普罗米修斯](#)
 - [clickhouse_exporter](#)
 - [PromHouse](#)
 - [clickhouse_exporter](#) (用途 [去客户](#))
 - [Nagios](#)
 - [check_clickhouse](#)
 - [check_clickhouse.py](#)
 - [Zabbix](#)
 - [Zabbix ClickHouse插件](#)
 - [Semantext](#)
 - [clickhouse积分](#)
- 记录
 - [rsyslog](#)
 - [Logstash ClickHouse输出](#)
 - [fluentd](#)
 - [loghouse](#) (对于 [Kubernetes](#))
 - [Semantext](#)
 - [Logstash ClickHouse输出插件](#)
- 地理
 - [MaxMind](#)
 - [GeoIP ClickHouse输出插件](#)

编程语言生态系统

- Python
 - [SQLAlchemy](#)
 - [clickhouse_orm](#) (使用 [infi.clickhouse_orm](#))
 - [能猫](#)

- pandahouse
- PHP
 - Doctrine
 - dbal-clickhouse
- R
 - dplyr
 - RClickhouse (使用 ツ暗エツ汎环催ツ団)
- Java
 - Hadoop
 - clickhouse-hdfs-装载机 (使用 JDBC)
- 斯卡拉
 - Akka
 - 捷胫client-禄晦鹿朐露胫鲁隆鹿-client酶
- C#
 - ADO.NET
 - 克莱克豪斯Ado
 - ClickHouse.Net
 - ClickHouse.Net.Migrations
- 仙丹
 - Ecto
 - clickhouse_ecto

C ++客户端库

请参阅以下网站的自述文件ツ暗エツ汎环催ツ団 资料库。

HTTP 客户端

HTTP 接口可以让你通过任何平台和编程语言来使用 ClickHouse。我们用 Java 和 Perl 以及 shell 脚本来访问它。在其他的部门中，HTTP 接口会用在 Perl，Python 以及 Go 中。HTTP 接口比 TCP 原生接口更为局限，但是却有更好的兼容性。

默认情况下，clickhouse-server 会在端口 8123 上监控 HTTP 请求（这可以在配置中修改）。

如果你发送了一个不带参数的 GET 请求，它会返回一个字符串 «Ok.» (结尾有换行)。可以将它用在健康检查脚本中。

```
$ curl 'http://localhost:8123/'  
Ok.
```

通过 URL 中的 `query` 参数来发送请求，或者发送 POST 请求，或者将查询的开头部分放在 URL 的 `query` 参数中，其他部分放在 POST 中（我们会在后面解释为什么这样做是有必要的）。URL 的大小会限制在 16 KB，所以发送大型查询时要时刻记住这点。

如果请求成功，将会收到 200 的响应状态码和响应主体中的结果。

如果发生了某个异常，将会收到 500 的响应状态码和响应主体中的异常描述信息。

当使用 GET 方法请求时，`readonly` 会被设置。换句话说，若要作修改数据的查询，只能发送 POST 方法的请求。可以将查询通过 POST 主体发送，也可以通过 URL 参数发送。

例：

```
$ curl 'http://localhost:8123/?query=SELECT%201'  
1  
  
$ wget -O- -q 'http://localhost:8123/?query=SELECT 1'  
1  
  
$ GET 'http://localhost:8123/?query=SELECT 1'
```

```
1
```

```
$ echo -ne 'GET /?query=SELECT%201 HTTP/1.0\r\n\r\n' | nc localhost 8123
HTTP/1.0 200 OK
Connection: Close
Date: Fri, 16 Nov 2012 19:21:50 GMT
```

```
1
```

可以看到，curl 命令由于空格需要 URL 转义，所以不是很方便。尽管 wget 命令对url做了 URL 转义，但我们并不推荐使用他，因为在 HTTP 1.1 协议下使用 keep-alive 和 Transfer-Encoding: chunked 头部设置它并不能很好的工作。

```
$ echo 'SELECT 1' | curl 'http://localhost:8123/' --data-binary @-
1

$ echo 'SELECT 1' | curl 'http://localhost:8123/?query=' --data-binary @-
1

$ echo '1' | curl 'http://localhost:8123/?query=SELECT' --data-binary @-
1
```

如果一部分请求是通过参数发送的，另外一部分通过 POST 主体发送，两部分查询之间会一行空行插入。

错误示例：

```
$ echo 'ECT 1' | curl 'http://localhost:8123/?query=SEL' --data-binary @-
Code: 59, e.displayText() = DB::Exception: Syntax error: failed at position 0: SEL
ECT 1
, expected One of: SHOW TABLES, SHOW DATABASES, SELECT, INSERT, CREATE, ATTACH, RENAME, DROP, DETACH,
USE, SET, OPTIMIZE., e.what() = DB::Exception
```

默认情况下，返回的数据是 TabSeparated 格式的，更多信息，见 «[数据格式]» 部分。

可以使用 FORMAT 设置查询来请求不同格式。

```
$ echo 'SELECT 1 FORMAT Pretty' | curl 'http://localhost:8123/?' --data-binary @-
```



INSERT 必须通过 POST 方法来插入数据。这种情况下，你可以将查询的开头部分放在 URL 参数中，然后用 POST 主体传入插入的数据。插入的数据可以是，举个例子，从 MySQL 导出的以 tab 分割的数据。在这种方式中，INSERT 查询取代了 LOAD DATA LOCAL INFILE from MySQL。

示例：创建一个表：

```
echo 'CREATE TABLE t (a UInt8) ENGINE = Memory' | POST 'http://localhost:8123/'
```

使用类似 INSERT 的查询来插入数据：

```
echo 'INSERT INTO t VALUES (1),(2),(3)' | POST 'http://localhost:8123/'
```

数据可以从查询中单独发送：

```
echo '(4),(5),(6)' | POST 'http://localhost:8123/?query=INSERT INTO t VALUES'
```

可以指定任何数据格式。值的格式和写入表 `t` 的值的格式相同：

```
echo '(7),(8),(9)' | POST 'http://localhost:8123/?query=INSERT INTO t FORMAT Values'
```

若要插入 `tab` 分割的数据，需要指定对应的格式：

```
echo -ne '10\n11\n12\n' | POST 'http://localhost:8123/?query=INSERT INTO t FORMAT TabSeparated'
```

从表中读取内容。由于查询处理是并行的，数据以随机顺序输出。

```
$ GET 'http://localhost:8123/?query=SELECT a FROM t'  
7  
8  
9  
10  
11  
12  
1  
2  
3  
4  
5  
6
```

删除表。

```
POST 'http://localhost:8123/?query=DROP TABLE t'
```

成功请求后并不会返回数据，返回一个空的响应体。

可以通过压缩来传输数据。压缩的数据没有一个标准的格式，但你需要指定一个压缩程序来使用它(`sudo apt-get install compressor-metrika-yandex`)。

如果在 URL 中指定了 `compress=1`，服务会返回压缩的数据。

如果在 URL 中指定了 `decompress=1`，服务会解压通过 POST 方法发送的数据。

可以通过为每份数据进行立即压缩来减少大规模数据传输中的网络压力。

可以指定 ‘`database`’ 参数来指定默认的数据库。

```
$ echo 'SELECT number FROM numbers LIMIT 10' | curl 'http://localhost:8123/?database=system' --data-binary @-  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

默认情况下，默认数据库会在服务的配置中注册，默认是 `default`。或者，也可以在表名之前使用一个点来指定数据库。

用户名密码可以通过以下两种方式指定：

1. 通过 HTTP Basic Authentication。示例：

```
echo 'SELECT 1' | curl 'http://user:password@localhost:8123/' -d @-
```

1. 通过 URL 参数中的 'user' 和 'password'。示例：

```
echo 'SELECT 1' | curl 'http://localhost:8123/?user=user&password=password' -d @-
```

如果用户名没有指定，默认的用户是 `default`。如果密码没有指定，默认会使用空密码。

可以使用 URL 参数指定配置或者设置整个配置文件来处理单个查询。示例：`http://localhost:8123/?profile=web&max_rows_to_read=1000000000&query=SELECT+1`

更多信息，参见《[设置](#)》部分。

```
$ echo 'SELECT number FROM system.numbers LIMIT 10' | curl 'http://localhost:8123/?' --data-binary @-
0
1
2
3
4
5
6
7
8
9
```

更多关于其他参数的信息，参见《[设置](#)》部分。

相比起 TCP 原生接口，HTTP 接口不支持会话和会话设置的概念，不允许中止查询（准确地说，只在少数情况下允许），不显示查询处理的进展。执行解析和数据格式化都是在服务端处理，网络上会比 TCP 原生接口更低效。

可选的 `query_id` 参数可能当做 query ID 传入（或者任何字符串）。更多信息，参见《[设置 replace_running_query](#)》部分。

可选的 `quota_key` 参数可能当做 quota key 传入（或者任何字符串）。更多信息，参见《[配额](#)》部分。

HTTP 接口允许传入额外的数据（外部临时表）来查询。更多信息，参见《[外部数据查询处理](#)》部分。

响应缓冲

可以在服务器端启用响应缓冲。提供了 `buffer_size` 和 `wait_end_of_query` 两个URL 参数来达此目的。

`buffer_size` 决定了查询结果要在服务内存中缓冲多少个字节数据。如果响应体比这个阈值大，缓冲区会写入到 HTTP 管道，剩下的数据也直接发到 HTTP 管道中。

为了确保整个响应体被缓冲，可以设置 `wait_end_of_query=1`。这种情况下，存入内存的数据会被缓冲到服务端的一个临时文件中。

示例：

```
curl -sS 'http://localhost:8123/?max_result_bytes=4000000&buffer_size=3000000&wait_end_of_query=1' -d 'SELECT
toUInt8(number) FROM system.numbers LIMIT 9000000 FORMAT RowBinary'
```

查询请求响应状态码和 HTTP 头被发送到客户端后，若发生查询处理出错，使用缓冲区可以避免这种情况的发生。在这种情况下，响应主体的结尾会写入一条错误消息，而在客户端，只能在解析阶段检测到该错误。

JDBC 驱动

- 官方 JDBC 的驱动
- 三方提供的 JDBC 驱动：
 - 捷径-禄海鹿脚露胫鲁隆鹿-酶
 - clickhouse4j

ODBC 驱动

- ClickHouse官方有 ODBC 的驱动。见 [这里](#)。

原生客户端接口 (TCP)

本机协议用于 [命令行客户端](#)，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。不幸的是，本机ClickHouse协议还没有正式的规范，但它可以从ClickHouse源代码进行逆向工程 [从这里开始](#)）和/或拦截和分析TCP流量。

命令行客户端

通过命令行来访问 ClickHouse，您可以使用 `clickhouse-client`

```
$ clickhouse-client
ClickHouse client version 0.0.26176.
Connecting to localhost:9000.
Connected to ClickHouse server version 0.0.26176.:)
```

该客户端支持命令行参数以及配置文件。查看更多，请看 [«配置»](#)

使用方式

这个客户端可以选择使用交互式与非交互式（批量）两种模式。

使用批量模式，要指定 `query` 参数，或者发送数据到 `stdin`（它会检查 `stdin` 是否是 Terminal），或者两种同时使用。它与 HTTP 接口很相似，当使用 `query` 参数发送数据到 `stdin` 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。

使用这个客户端插入数据的示例：

```
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";

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

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

在批量模式中，默认的数据格式是 `TabSeparated` 分隔的。您可以根据查询来灵活设置 `FORMAT` 格式。

默认情况下，在批量模式中只能执行单个查询。为了从一个 Script 中执行多个查询，可以使用 `--multิquery` 参数。除了 `INSERT` 请求外，这种方式在任何地方都有用。查询的结果会连续且不含分隔符地输出。

同样的，为了执行大规模的查询，您可以为每个查询执行一次 `clickhouse-client`。但注意到每次启动 `clickhouse-client` 程序都需要消耗几十毫秒时间。

在交互模式下，每条查询过后，你可以直接输入下一条查询命令。

如果 `multiline` 没有指定（默认没指定）：为了执行查询，按下 `Enter` 即可。查询语句不是必须使用分号结尾。如果需要写一个多个行的查询语句，可以在换行之前输入一个反斜杠`\`，然后在您按下 `Enter` 键后，您就可以输入当前语句的下一行查询了。

如果 `multiline` 指定了：为了执行查询，需要以分号结尾并且按下 `Enter` 键。如果行末没有分号，将认为当前语句并没有输入完而要求继续输入下一行。

若只运行单个查询，分号后面的所有内容都会被忽略。

您可以指定 `\G` 来替代分号或者在分号后面，这表示 `Vertical` 的格式。在这种格式下，每一个值都会打印在不同的行中，这种方式对于宽表来说很方便。这个不常见的特性是为了兼容 MySQL 命令而加的。

命令行客户端是基于 `replxx`。换句话说，它可以使用我们熟悉的快捷键方式来操作以及保留历史命令。

历史命令会写入在 `~/.clickhouse-client-history` 中。

默认情况下，输出的格式是 `PrettyCompact`。您可以通过 `FORMAT` 设置根据不同查询来修改格式，或者通过在查询末尾指定 `\G` 字符，或通过在命令行中使用 `--format` 或 `--vertical` 参数，或使用客户端的配置文件。

若要退出客户端，使用 `Ctrl+D`（或 `Ctrl+C`），或者输入以下其中一个命令：`exit`, `quit`, `logout`, `учше`, `йгше`, `дщпшге`, `exit;`, `quit;`, `logout;`, `учшеж`, `йгшеж`, `дщпшгеж`, `q`, `й`, `q`, `Q`, `:q`, `й`, `Й`, `ЖЙ`

当执行一个查询的时候，客户端会显示：

1. 进度，进度会每秒更新十次（默认情况下）。对于很快的查询，进度可能没有时间显示。
2. 为了调试会显示解析且格式化后的查询语句。
3. 指定格式的输出结果。
4. 输出结果的行数的行数，经过的时间，以及查询处理的速度。

您可以通过 `Ctrl+C` 来取消一个长时间的查询。然而，您依然需要等待服务端来中止请求。在某个阶段去取消查询是不可能的。如果您不等待并再次按下 `Ctrl + C`，客户端将会退出。

命令行客户端允许通过外部数据（外部临时表）来查询。更多相关信息，请参考《[外部数据查询处理](#)》。

配置

您可以通过以下方式传入参数到 `clickhouse-client` 中（所有的参数都有默认值）：

- 通过命令行

命令行参数会覆盖默认值和配置文件的配置。

- 配置文件

配置文件的配置会覆盖默认值

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 原生接口是使用不同端口的。
- `--user, -u` – 用户名。默认值： default。
- `--password` – 密码。默认值：空字符串。
- `--query, -q` – 非交互模式下的查询语句。
- `--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，允许处理用逗号分隔的多个查询，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。

配置文件

`clickhouse-client` 使用一下第一个左的文件。

- 通过 `--config-file` 参数指定的文件.
- `./clickhouse-client.xml`
- `\~/.clickhouse-client/config.xml`
- `/etc/clickhouse-client/config.xml`

配置文件示例:

```
<config>
  <user>username</user>
  <password>password</password>
</config>
```

客户端

ClickHouse提供了两个网络接口（两者都可以选择包装在TLS中以提高安全性）：

- **HTTP**，记录在案，易于使用.
- **本地TCP**，这有较少的开销.

在大多数情况下，建议使用适当的工具或库，而不是直接与这些工具或库进行交互。Yandex的官方支持如下：

- * 命令行客户端
- * JDBC驱动程序
- * ODBC驱动程序
- * C++客户端库

还有许多第三方库可供使用ClickHouse：

- * 客户端库
- * 集成
- * 可视界面

输入输出格式

ClickHouse 可以接受多种数据格式，可以在 (INSERT) 以及 (SELECT) 请求中使用。

下列表格列出了支持的数据格式以及在 (INSERT) 以及 (SELECT) 请求中使用它们的方式。

格式	INSERT	SELECT
TabSeparated	✓	✓
TabSeparatedRaw	✗	✓
TabSeparatedWithNames	✓	✓
TabSeparatedWithNamesAndTypes	✓	✓
模板	✓	✓
TemplateIgnoreSpaces	✓	✗
CSV	✓	✓
CSVWithNames	✓	✓
自定义分离	✓	✓

格式	INSERT	SELECT
值	✓	✓
垂直	✗	✓
VerticalRaw	✗	✓
JSON	✗	✓
JSONCompact	✗	✓
JSONEachRow	✓	✓
TSKV	✓	✓
漂亮	✗	✓
PrettyCompact	✗	✓
PrettyCompactMonoBlock	✗	✓
PrettyNoEscapes	✗	✓
PrettySpace	✗	✓
Protobuf	✓	✓
Avro	✓	✓
AvroConfluent	✓	✗
木地板	✓	✓
ORC	✓	✗
RowBinary	✓	✓
RowBinaryWithNamesAndTypes	✓	✓
本地人	✓	✓
Null	✗	✓
XML	✗	✓
CapnProto	✓	✓

TabSeparated

在 TabSeparated 格式中，数据按行写入。每行包含由制表符分隔的值。除了行中的最后一个值（后面紧跟换行符）之外，每个值都跟随一个制表符。在任何地方都可以使用严格的 Unix 命令行。最后一行还必须在最后包含换行符。值以文本格式编写，不包含引号，并且要转义特殊字符。

这种格式也可以用 `TSV` 来表示。

TabSeparated 格式非常方便用于自定义程序或脚本处理数据。HTTP 客户端接口默认会用这种格式，命令行客户端批量模式下也会用这种格式。这种格式允许在不同数据库之间传输数据。例如，从 MySQL 中导出数据然后导入到 ClickHouse 中，反之亦然。

`TabSeparated` 格式支持输出数据总值（当使用 `WITH TOTALS`）以及极值（当 ‘extremes’ 设置是1）。这种情况下，总值和极值输出在主数据的后面。主要的数据，总值，极值会以一个空行隔开，例如：

```
SELECT EventDate, count() AS c FROM test.hits GROUP BY EventDate WITH TOTALS ORDER BY EventDate FORMAT TabSeparated``
```

2014-03-17	1406958
2014-03-18	1383658
2014-03-19	1405797
2014-03-20	1353623
2014-03-21	1245779
2014-03-22	1031592
2014-03-23	1046491
0000-00-00	8873898
2014-03-17	1031592
2014-03-23	1406958

数据解析方式

整数以十进制形式写入。数字在开头可以包含额外的 + 字符（解析时忽略，格式化时不记录）。非负数不能包含负号。读取时，允许将空字符串解析为零，或者（对于带符号的类型）将仅包含负号的字符串解析为零。不符合相应数据类型的数字可能会被解析为不同的数字，而不会显示错误消息。

浮点数以十进制形式写入。点号用作小数点分隔符。支持指数等符号，如'inf'，'+ inf'，'-inf'和'nan'。浮点数的输入可以以小数点开始或结束。

格式化的时候，浮点数的精确度可能会丢失。

解析的时候，没有严格需要去读取与机器可以表示的最接近的数值。

日期会以 YYYY-MM-DD 格式写入和解析，但会以任何字符作为分隔符。

带时间的日期会以 YYYY-MM-DD hh:mm:ss 格式写入和解析，但会以任何字符作为分隔符。

这一切都发生在客户端或服务器启动时的系统时区（取决于哪一种格式的数据）。对于具有时间的日期，夏时制时间未指定。因此，如果转储在夏令时中有时间，则转储不会明确地匹配数据，解析将选择两者之一。

在读取操作期间，不正确的日期和具有时间的日期可以使用自然溢出或空日期和时间进行分析，而不会出现错误消息。

有个例外情况，Unix 时间戳格式（10个十进制数字）也支持使用时间解析日期。结果不是时区相关的。格式 YYYY-MM-DD hh:mm:ss 和 NNNNNNNNNN 会自动区分。

字符串以反斜线转义的特殊字符输出。以下转义序列用于输出：`\b`, `\f`, `\r`, `\n`, `\t`, `\0`, `'`, `\"`。解析还支持`\a`, `\v`和`\xHH`（十六进制转义字符）和任何`\c`字符，其中`c`是任何字符（这些序列被转换为`c`）。因此，读取数据支持可以将换行符写为`\n`或`\r\n`的格式，或者换行。例如，字符串`Hello world`在单词之间换行而不是空格可以解析为以下任何形式：

Hello\\nworld

Hello\\
world

第二种形式是支持的，因为 MySQL 读取 tab-separated 格式数据集的时候也会使用它。

在 TabSeparated 格式中传递数据时需要转义的最小字符集为：Tab，换行符（LF）和反斜杠。

只有一小组符号会被转义。你可以轻易地找到一个字符串值，但这不会正常在你的终端显示。

数组写在方括号内的逗号分隔值列表中。通常情况下，数组中的数字项目会被拼凑，但日期，带时间的日期以及字符串将使用与上面相同的转义规则用单引号引起来。

NULL 将输出为 `\N`。

TabSeparatedRaw

与 `TabSeparated` 格式不一样的是，行数据是不会被转义的。

该格式仅适用于输出查询结果，但不适用于解析输入（将数据插入到表中）。

这种格式也可以使用名称 `TSVRaw` 来表示。

TabSeparatedWithNames

与 `TabSeparated` 格式不一样的是，第一行会显示列的名称。

在解析过程中，第一行完全被忽略。您不能使用列名来确定其位置或检查其正确性。

（未来可能会加入解析头行的功能）

这种格式也可以使用名称 `TSVWithNames` 来表示。

TabSeparatedWithNamesAndTypes

与 `TabSeparated` 格式不一样的是，第一行会显示列的名称，第二行会显示列的类型。

在解析过程中，第一行和第二行完全被忽略。

这种格式也可以使用名称 `TSVWithNamesAndTypes` 来表示。

模板

此格式允许为具有指定转义规则的值指定带有占位符的自定义格式字符串。

它使用设置 `format_schema`, `format_schema_rows`, `format_schema_rows_between_delimiter` and some settings of other formats (e.g. `output_format_json_quote_64bit_integers` 使用时 `JSON` 逃跑，进一步查看)

格式字符串 `format_schema_rows` 使用以下语法指定行格式:

```
delimiter_1${column_1:serializeAs_1}delimiter_2${column_2:serializeAs_2} ... delimiter_N,
```

where `delimiter_i` is a delimiter between values (`\$` symbol can be escaped as `\$\$`),
`column_i` is a name of a column whose values are to be selected or inserted (if empty, then column will be skipped),
`serializeAs_i` is an escaping rule for the column values. The following escaping rules are supported:

- `CSV` , `JSON` , `XML` (similarly to the formats of the same names)
- `Escaped` (similarly to `TSV`)
- `Quoted` (similarly to `Values`)
- `Raw` (without escaping, similarly to `TSVRaw`)
- `None` (no escaping rule, see further)

If escaping rule is omitted, then `None` will be used. `XML` and `Raw` are suitable only for output.

So, for the following format string:

```
`Search phrase: ${SearchPhrase:Quoted}, count: ${c:Escaped}, ad price: $$ ${price:JSON};`
```

the values of `SearchPhrase` , `c` and `price` columns, which are escaped as `Quoted` , `Escaped` and `JSON` will be printed (for select) or will be expected (for insert) between `Search phrase: ` , ` , count: ` , ` , ad price: \$` and `;` delimiters respectively. For example:

```
`Search phrase: 'bathroom interior design', count: 2166, ad price: $3;`
```

该 `format_schema_rows_between_delimiter` setting 指定行之间的分隔符，该分隔符在除最后一行之外的每一行之后打印（或预期）（`\n` 默认情况下）

格式字符串 `format_schema` 具有相同的语法 `format_schema_rows` 并允许指定前缀，后缀和打印一些附加信息的方式。它包含以下占位符而不是列名：

- `data` 包含数据的行 `format_schema_rows` 格式，由分隔 `format_schema_rows_between_delimiter`. 此占位符必须是格式字符串中的第一个占位符。
- `totals` 是包含总值的行 `format_schema_rows` 格式（与总计一起使用时）
- `min` 是具有最小值的行 `format_schema_rows` 格式（当极值设置为1时）
- `max` 是具有最大值的行 `format_schema_rows` 格式（当极值设置为1时）
- `rows` 输出行总数
- `rows_before_limit` 是没有限制的最小行数。仅当查询包含LIMIT时输出。如果查询包含GROUP BY，则 `rows_before_limit_at_least` 是没有限制的确切行数。
- `time` 请求执行时间以秒为单位
- `rows_read` 已读取的行数
- `bytes_read` 被读取的字节数（未压缩）

占位符 `data`, `totals`, `min` 和 `max` 必须没有指定转义规则（或 `None` 必须明确指定）。其余的占位符可能具有指定的任何转义规则。

如果 `format_schema` 设置为空字符串，`#{data}` 用作默认值。

对于插入查询格式允许跳过一些列或一些字段，如果前缀或后缀（见示例）。

Select 示例：

```
SELECT SearchPhrase, count() AS c FROM test.hits GROUP BY SearchPhrase ORDER BY c DESC LIMIT 5
FORMAT Template
SETTINGS format_schema = '<!DOCTYPE HTML>
<html> <head> <title>Search phrases</title> </head>
<body>
<table border="1"> <caption>Search phrases</caption>
<tr> <th>Search phrase</th> <th>Count</th> </tr>
${data}
</table>
<table border="1"> <caption>Max</caption>
${max}
</table>
<b>Processed ${rows_read:XML} rows in ${time:XML} sec</b>
</body>
</html>',
format_schema_rows = '<tr> <td>${SearchPhrase:XML}</td> <td>${c:XML}</td> </tr>',
format_schema_rows_between_delimiter = '\n  '
```

```
<!DOCTYPE HTML>
<html> <head> <title>Search phrases</title> </head>
<body>
<table border="1"> <caption>Search phrases</caption>
<tr> <th>Search phrase</th> <th>Count</th> </tr>
<tr> <td></td> <td>8267016</td> </tr>
<tr> <td>bathroom interior design</td> <td>2166</td> </tr>
<tr> <td>yandex</td> <td>1655</td> </tr>
<tr> <td>spring 2014 fashion</td> <td>1549</td> </tr>
<tr> <td>freeform photos</td> <td>1480</td> </tr>
</table>
<table border="1"> <caption>Max</caption>
<tr> <td></td> <td>8873898</td> </tr>
</table>
<b>Processed 3095973 rows in 0.1569913 sec</b>
```

```
</body>
</html>
```

Insert 示例：

```
Some header
Page views: 5, User id: 4324182021466249494, Useless field: hello, Duration: 146, Sign: -1
Page views: 6, User id: 4324182021466249494, Useless field: world, Duration: 185, Sign: 1
Total rows: 2
```

```
INSERT INTO UserActivity FORMAT Template SETTINGS
format_schema = 'Some header\n${data}\nTotal rows: ${:CSV}\n',
format_schema_rows = 'Page views: ${PageViews:CSV}, User id: ${UserID:CSV}, Useless field: ${:CSV}, Duration: ${Duration:CSV}, Sign: ${Sign:CSV}'
```

`PageViews`, `UserID`, `Duration` 和 `Sign` 占位符内部是表中列的名称。值后 `Useless field` 在行和之后 `\nTotal rows:` 后缀将被忽略。

输入数据中的所有分隔符必须严格等于指定格式字符串中的分隔符。

TemplateIgnoreSpaces

此格式仅适用于输入。

类似于 `Template`，但跳过输入流中的分隔符和值之间的空格字符。但是，如果格式字符串包含空格字符，则在输入流中将需要这些字符。还允许指定空白占位符 (`{}$` 或 `{}{:None}`) 将一些分隔符分成单独的部分，以忽略它们之间的空格。此类占位符仅用于跳过空格字符。

可以阅读 `JSON` 如果列的值在所有行中具有相同的顺序，则使用此格式。例如，以下请求可用于从格式的输出示例中插入数据 `JSON`：

```
INSERT INTO table_name FORMAT TemplateIgnoreSpaces SETTINGS
format_schema = '{${"meta":${}:${:JSON},${}"data":${}:${}
[${"data"}${},${}"totals":${}:${:JSON},${}"extremes":${}:${:JSON},${}"rows":${}:${:JSON},${}"rows_before_limit_at
_least":${}:${:JSON}${}}}', 
format_schema_rows = '{${"SearchPhrase":${}:${}{$phrase:JSON}${},${}"c":${}:${}{$cnt:JSON}${}}}', 
format_schema_rows_between_delimiter = ','
```

TSKV

与 `TabSeparated` 格式类似，但它输出的是 `name=value` 的格式。名称会和 `TabSeparated` 格式一样被转义，`=` 字符也会被转义。

```
SearchPhrase= count()=8267016
SearchPhrase=bathroom interior design count()=2166
SearchPhrase=yandex count()=1655
SearchPhrase=2014 spring fashion count()=1549
SearchPhrase=freeform photos count()=1480
SearchPhrase=angelina jolie count()=1245
SearchPhrase=omsk count()=1112
SearchPhrase=photos of dog breeds count()=1091
SearchPhrase=curtain designs count()=1064
SearchPhrase=baku count()=1000
```

`NULL` 输出为 `\N`。

```
SELECT * FROM t_null FORMAT TSKV
```

```
x=1 y=\N
```

当有大量的小列时，这种格式是低效的，通常没有理由使用它。它被用于 Yandex 公司的一些部门。

数据的输出和解析都支持这种格式。对于解析，任何顺序都支持不同列的值。可以省略某些值，用 `-` 表示，它们被视为等于它们的默认值。在这种情况下，零和空行被用作默认值。作为默认值，不支持表中指定的复杂值。

对于不带等号或值，可以用附加字段 `tskv` 来表示，这种在解析上是被允许的。这样的话该字段被忽略。

CSV

按逗号分隔的数据格式([RFC](#))。

格式化的时候，行是用双引号括起来的。字符串中的双引号会以两个双引号输出，除此之外没有其他规则来做字符转义了。日期和时间也会以双引号包括。数字的输出不带引号。值由一个单独的字符隔开，这个字符默认是 `,`。行使用 Unix 换行符 (`LF`) 分隔。数组序列化成 CSV 规则如下：首先将数组序列化为 TabSeparated 格式的字符串，然后将结果字符串用双引号包括输出到 CSV。CSV 格式的元组被序列化为单独的列（即它们在元组中的嵌套关系会丢失）。

```
clickhouse-client --format_csv_delimiter="|" --query="INSERT INTO test.csv FORMAT CSV" < data.csv
```

*默认情况下间隔符是 `,`，在 `format_csv_delimiter` 中可以了解更多间隔符配置。

解析的时候，可以使用或不使用引号来解析所有值。支持双引号和单引号。行也可以不用引号排列。在这种情况下，它们被解析为逗号或换行符 (CR 或 LF)。在解析不带引号的行时，若违反 RFC 规则，会忽略前导和尾随的空格和制表符。对于换行，全部支持 Unix (LF)，Windows (CR LF) 和 Mac OS Classic (CR LF)。

`NULL` 将输出为 `\N`。

CSV 格式是和 TabSeparated 一样的方式输出总数和极值。

CSVWithNames

会输出带头部行，和 `TabSeparatedWithNames` 一样。

自定义分离

类似于 [模板](#)，但它打印或读取所有列，并使用从设置转义规则 `format_custom_escaping_rule` 从设置和分隔符 `format_custom_field_delimiter`, `format_custom_row_before_delimiter`, `format_custom_row_after_delimiter`, `format_custom_row_between_delimiter`, `format_custom_result_before_delimiter` 和 `format_custom_result_after_delimiter`，而不是从格式字符串。

也有 `CustomSeparatedIgnoreSpaces` 格式，这是类似于 `TemplateIgnoreSpaces`。

JSON

以 JSON 格式输出数据。除了数据表之外，它还输出列名称和类型以及一些附加信息：输出行的总数以及在没有 LIMIT 时可以输出的行数。例：

```
SELECT SearchPhrase, count() AS c FROM test.hits GROUP BY SearchPhrase WITH TOTALS ORDER BY c DESC LIMIT 5
FORMAT JSON
```

```
{
  "meta": [
    {
      "name": "SearchPhrase",
      "type": "String"
    }
  ]
}
```

```

        },
        {
            "name": "c",
            "type": "UInt64"
        }
    ],
    "data":
    [
        {
            "SearchPhrase": "",
            "c": "8267016"
        },
        {
            "SearchPhrase": "bathroom interior design",
            "c": "2166"
        },
        {
            "SearchPhrase": "yandex",
            "c": "1655"
        },
        {
            "SearchPhrase": "spring 2014 fashion",
            "c": "1549"
        },
        {
            "SearchPhrase": "freeform photos",
            "c": "1480"
        }
    ],
    "totals":
    {
        "SearchPhrase": "",
        "c": "8873898"
    },
    "extremes":
    {
        "min":
        {
            "SearchPhrase": "",
            "c": "1480"
        },
        "max":
        {
            "SearchPhrase": "",
            "c": "8267016"
        }
    },
    "rows": 5,
    "rows_before_limit_at_least": 141137
}

```

JSON 与 JavaScript 兼容。为了确保这一点，一些字符被另外转义：斜线 / 被转义为 \ /; 替代的换行符 U+2028 和 U+2029 会打断一些浏览器解析，它们会被转义为 \uXXXX。ASCII 控制字符被转义：退格，换页，换行，回车和水平制表符被替换为 \b，\f，\n，\r，\t 作为使用 \uXXXX 序列的 00-1F 范围内的剩余字节。无效的 UTF-8 序列更改为替换字符，因此输出文本将包含有效的 UTF-8 序列。为了与 JavaScript 兼容，默认情况下，Int64 和 UInt64 整数用双引号引起来。要除去引号，可以将配置参数 output_format_json_quote_64bit_integers 设置为 0。

`rows` – 结果输出的行数。

`rows_before_limit_at_least` 去掉 LIMIT 过滤后的最小行总数。只会在查询包含 LIMIT 条件时输出。

若查询包含 GROUP BY，`rows_before_limit_at_least` 就是去掉 LIMIT 后过滤后的准确行数。

`totals` – 总值（当使用 `TOTALS` 条件时）。

`extremes` – 极值（当 `extremes` 设置为 1 时）。

该格式仅适用于输出查询结果，但不适用于解析输入（将数据插入到表中）。

ClickHouse 支持 `NULL`，在 JSON 格式中以 `null` 输出来表示。

参考 `JSONEachRow` 格式。

JSONCompact

与 JSON 格式不同的是它以数组的方式输出结果，而不是以结构体。

示例：

```
{
    "meta": [
        {
            "name": "SearchPhrase",
            "type": "String"
        },
        {
            "name": "c",
            "type": "UInt64"
        }
    ],
    "data": [
        ["", "8267016"],
        ["bathroom interior design", "2166"],
        ["yandex", "1655"],
        ["fashion trends spring 2014", "1549"],
        ["freeform photo", "1480"]
    ],
    "totals": ["", "8873898"],
    "extremes": {
        "min": ["", "1480"],
        "max": ["", "8267016"]
    },
    "rows": 5,
    "rows_before_limit_at_least": 141137
}
```

这种格式仅仅适用于输出结果集，而不适用于解析（将数据插入到表中）。

参考 `JSONEachRow` 格式。

JSONEachRow

将数据结果每一行以 JSON 结构体输出（换行分割 JSON 结构体）。

```
{"SearchPhrase":"","count()":"8267016"}  
{"SearchPhrase": "bathroom interior design","count()": "2166"}  
{"SearchPhrase": "yandex","count()": "1655"}  
{"SearchPhrase": "2014 spring fashion","count()": "1549"}  
{"SearchPhrase": "freeform photo","count()": "1480"}  
{"SearchPhrase": "angelina jolie","count()": "1245"}  
{"SearchPhrase": "omsk","count()": "1112"}  
{"SearchPhrase": "photos of dog breeds","count()": "1091"}  
{"SearchPhrase": "curtain designs","count()": "1064"}  
{"SearchPhrase": "baku","count()": "1000"}
```

与 JSON 格式不同的是，没有替换无效的UTF-8序列。任何一组字节都可以在行中输出。这是必要的，因为这样数据可以被格式化而不会丢失任何信息。值的转义方式与JSON相同。

对于解析，任何顺序都支持不同列的值。可以省略某些值 - 它们被视为等于它们的默认值。在这种情况下，零和空行被用作默认值。作为默认值，不支持表中指定的复杂值。元素之间的空白字符被忽略。如果在对象之后放置逗号，它将被忽略。对象不一定必须用新行分隔。

嵌套结构的使用

如果你有一张桌子 嵌套式 数据类型列，可以插入具有相同结构的JSON数据。启用此功能与 `input_format_import_nested_json` 设置。

例如，请考虑下表：

```
CREATE TABLE json_each_row_nested (n Nested (s String, i Int32) ) ENGINE = Memory
```

正如你可以在找到 `Nested` 数据类型说明，ClickHouse将嵌套结构的每个组件视为单独的列，`n.s` 和 `n.i` 为了我们的桌子 所以你可以通过以下方式插入数据：

```
INSERT INTO json_each_row_nested FORMAT JSONEachRow {"n.s": ["abc", "def"], "n.i": [1, 23]}
```

将数据作为分层JSON对象集插入 `input_format_import_nested_json=1`.

```
{  
  "n": {  
    "s": ["abc", "def"],  
    "i": [1, 23]  
  }  
}
```

如果没有此设置，ClickHouse将引发异常。

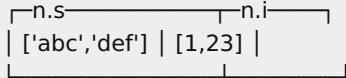
```
SELECT name, value FROM system.settings WHERE name = 'input_format_import_nested_json'
```

name	value
input_format_import_nested_json	0

```
INSERT INTO json_each_row_nested FORMAT JSONEachRow {"n": {"s": ["abc", "def"], "i": [1, 23]}}
```

```
Code: 117. DB::Exception: Unknown field found while parsing JSONEachRow format: n: (at row 1)
```

```
SET input_format_import_nested_json=1
INSERT INTO json_each_row_nested FORMAT JSONEachRow {"n": {"s": ["abc", "def"], "i": [1, 23]}}
SELECT * FROM json_each_row_nested
```



本地人

最高性能的格式。据通过二进制格式的块进行写入和读取。对于每个块，该块中的行数，列数，列名称和类型以及列的部分将被相继记录。换句话说，这种格式是《列式》的 - 它不会将列转换为行。这是用于在服务器之间进行交互的本地界面中使用的格式，用于使用命令行客户端和 C++ 客户端。

您可以使用此格式快速生成只能由 ClickHouse DBMS 读取的格式。但自己处理这种格式是没有意义的。

Null

没有输出。但是，查询已处理完毕，并且在使用命令行客户端时，数据将传输到客户端。这仅用于测试，包括生产力测试。显然，这种格式只适用于输出，不适用于解析。

漂亮

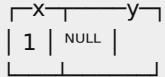
将数据以表格形式输出，也可以使用 ANSI 转义字符在终端中设置颜色。

它会绘制一个完整的表格，每行数据在终端中占用两行。

每一个结果块都会以单独的表格输出。这是很有必要的，以便结果块不用缓冲结果输出（缓冲在可以预见结果集宽度的时候是很有必要的）。

NULL 输出为 `NULL`。

```
SELECT * FROM t_null
```



为避免将太多数据传输到终端，只打印前10,000行。如果行数大于或等于10,000，则会显示消息«Showed first 10 000»。

该格式仅适用于输出查询结果，但不适用于解析输入（将数据插入到表中）。

Pretty格式支持输出总值（当使用 `WITH TOTALS` 时）和极值（当 `extremes` 设置为1时）。在这些情况下，总数值和极值在主数据之后以单独的表格形式输出。示例（以 PrettyCompact 格式显示）：

```
SELECT EventDate, count() AS c FROM test.hits GROUP BY EventDate WITH TOTALS ORDER BY EventDate FORMAT PrettyCompact
```

EventDate	c
2014-03-17	1406958
2014-03-18	1383658
2014-03-19	1405797
2014-03-20	1353623

2014-03-21	1245779
2014-03-22	1031592
2014-03-23	1046491

Totals:

EventDate	c
0000-00-00	8873898

Extremes:

EventDate	c
2014-03-17	1031592
2014-03-23	1406958

PrettyCompact

与 `Pretty` 格式不一样的是，`PrettyCompact` 去掉了行之间的表格分割线，这样使得结果更加紧凑。这种格式会在交互命令行客户端下默认使用。

PrettyCompactMonoBlock

与 `PrettyCompact` 格式不一样的是，它支持 10,000 行数据缓冲，然后输出在一个表格中，不会按照块来区分

PrettyNoEscapes

与 `Pretty` 格式不一样的是，它不使用 ANSI 字符转义，这在浏览器显示数据以及在使用 `watch` 命令行工具是有必要的。

示例：

```
watch -n1 "clickhouse-client --query='SELECT event, value FROM system.events FORMAT PrettyCompactNoEscapes'"
```

您可以使用 HTTP 接口来获取数据，显示在浏览器中。

PrettyCompactNoEscapes

用法类似上述。

PrettySpaceNoEscapes

用法类似上述。

PrettySpace

与 `PrettyCompact`(#prettycompact) 格式不一样的是，它使用空格来代替网格来显示数据。

RowBinary

以二进制格式逐行格式化和解析数据。行和值连续列出，没有分隔符。

这种格式比 Native 格式效率低，因为它是基于行的。

整数使用固定长度的小端表示法。例如，UInt64 使用8个字节。

DateTime 被表示为 UInt32 类型的Unix 时间戳值。

Date 被表示为 UInt16 对象，它的值为 1970-01-01 以来的天数。

字符串表示为 varint 长度（无符号 LEB128），后跟字符串的字节数。

FixedString 被简单地表示为一个字节序列。

数组表示为 varint 长度（无符号 LEB128），后跟有序的数组元素。

对于 NULL 的支持，一个为 1 或 0 的字节会加在每个 可为空 值前面。如果为 1，那么该值就是 NULL。如果为 0，则不为

`NULL`。

RowBinaryWithNamesAndTypes

类似于 `RowBinary`，但添加了标题：

- `LEB128`-编码列数 (`N`)
- `N String`s 指定列名
- `N String`s 指定列类型

值

在括号中打印每一行。行由逗号分隔。最后一行之后没有逗号。括号内的值也用逗号分隔。数字以十进制格式输出，不含引号。数组以方括号输出。带有时间的字符串，日期和时间用引号包围输出。转义字符的解析规则与 `TabSeparated` 格式类似。在格式化过程中，不插入额外的空格，但在解析过程中，空格是被允许并跳过的（除了数组值之外的空格，这是不允许的）。`NULL` 为 `NULL`。

以 `Values` 格式传递数据时需要转义的最小字符集是：单引号和反斜线。

这是 `INSERT INTO t VALUES ...` 中可以使用的格式，但您也可以将其用于查询结果。

垂直

使用指定的列名在单独的行上打印每个值。如果每行都包含大量列，则此格式便于打印一行或几行。

`NULL` 输出为 `NULL`。

示例：

```
SELECT * FROM t_null FORMAT Vertical
```

Row 1:

```
x: 1  
y: NULL
```

该格式仅适用于输出查询结果，但不适用于解析输入（将数据插入到表中）。

VerticalRaw

和 `Vertical` 格式不同点在于，行是不会被转义的。

这种格式仅仅适用于输出，但不适用于解析输入（将数据插入到表中）。

示例：

```
:) SHOW CREATE TABLE geonames FORMAT VerticalRaw;  
Row 1:  
  
statement: CREATE TABLE default.geonames ( geonameid UInt32, date Date DEFAULT CAST('2017-12-08' AS Date))  
ENGINE = MergeTree(date, geonameid, 8192)  
  
:) SELECT 'string with \'quotes\' and \t with some special \n characters' AS test FORMAT VerticalRaw;  
Row 1:  
  
test: string with 'quotes' and  with some special  
characters
```

和 `Vertical` 格式相比：

```
:) SELECT 'string with \'quotes\' and \t with some special \n characters' AS test FORMAT Vertical;
```

```
Row 1:
```

```
test: string with \'quotes\' and \t with some special \n characters
```

XML

该格式仅适用于输出查询结果，但不适用于解析输入，示例：

```
<?xml version='1.0' encoding='UTF-8' ?>
<result>
  <meta>
    <columns>
      <column>
        <name>SearchPhrase</name>
        <type>String</type>
      </column>
      <column>
        <name>count()</name>
        <type>UInt64</type>
      </column>
    </columns>
  </meta>
  <data>
    <row>
      <SearchPhrase></SearchPhrase>
      <field>8267016</field>
    </row>
    <row>
      <SearchPhrase>bathroom interior design</SearchPhrase>
      <field>2166</field>
    </row>
    <row>
      <SearchPhrase>yandex</SearchPhrase>
      <field>1655</field>
    </row>
    <row>
      <SearchPhrase>2014 spring fashion</SearchPhrase>
      <field>1549</field>
    </row>
    <row>
      <SearchPhrase>freeform photos</SearchPhrase>
      <field>1480</field>
    </row>
    <row>
      <SearchPhrase>angelina jolie</SearchPhrase>
      <field>1245</field>
    </row>
    <row>
      <SearchPhrase>omsk</SearchPhrase>
      <field>1112</field>
    </row>
    <row>
      <SearchPhrase>photos of dog breeds</SearchPhrase>
      <field>1091</field>
    </row>
    <row>
      <SearchPhrase>curtain designs</SearchPhrase>
      <field>1064</field>
    </row>
  </data>
</result>
```

```
<row>
  <SearchPhrase>baku</SearchPhrase>
  <field>1000</field>
</row>
</data>
<rows>10</rows>
<rows_before_limit_at_least>141137</rows_before_limit_at_least>
</result>
```

如果列名称没有可接受的格式，则仅使用 `field` 作为元素名称。通常，XML 结构遵循 JSON 结构。就像 JSON 一样，将无效的 UTF-8 字符都作替换，以便输出文本将包含有效的 UTF-8 字符序列。

在字符串值中，字符 `<` 和 `&` 被转义为 `<` 和 `&`。

数组输出为 `<array> <elem> Hello </ elem> <elem> World </ elem> ... </ array>`，元组输出为 `<tuple> <elem> Hello </ elem> <elem> World </ ELEM> ... </tuple>`。

CapnProto

Cap'n Proto 是一种二进制消息格式，类似 Protocol Buffers 和 Thrift，但与 JSON 或 MessagePack 格式不一样。

Cap'n Proto 消息格式是严格类型的，而不是自我描述，这意味着它们不需要外部的描述。这种格式可以实时地应用，并针对每个查询进行缓存。

```
SELECT SearchPhrase, count() AS c FROM test.hits
  GROUP BY SearchPhrase FORMAT CapnProto SETTINGS schema = 'schema:Message'
```

其中 `schema.capnp` 描述如下：

```
struct Message {
  SearchPhrase @0 :Text;
  c @1 :Uint64;
}
```

格式文件存储的目录可以在服务配置中的 `format_schema_path` 指定。

Cap'n Proto 反序列化是很高效的，通常不会增加系统的负载。

Protobuf

Protobuf 是一个 [协议缓冲区](#) 格式。

此格式需要外部格式架构。在查询之间缓存架构。

ClickHouse 支持 `proto2` 和 `proto3` 语法 支持重复/可选/必填字段。

使用示例：

```
SELECT * FROM test.table FORMAT Protobuf SETTINGS format_schema = 'schemafile:MessageType'
```

```
cat protobuf_messages.bin | clickhouse-client --query "INSERT INTO test.table FORMAT Protobuf SETTINGS
format_schema='schemafile:MessageType'"
```

哪里的文件 `schemafile.proto` 看起来像这样：

```
syntax = "proto3";
```

```

message MessageType {
    string name = 1;
    string surname = 2;
    uint32 birthDate = 3;
    repeated string phoneNumbers = 4;
};

```

要查找协议缓冲区的消息类型的表列和字段之间的对应关系，ClickHouse比较它们的名称。

这种比较是不区分大小写和字符 `_` (下划线)和 `.` (点) 被认为是相等的。

如果协议缓冲区消息的列和字段的类型不同，则应用必要的转换。

支持嵌套消息。例如，对于字段 `z` 在下面的消息类型

```

message MessageType {
    message XType {
        message YType {
            int32 z;
        };
        repeated YType y;
    };
    XType x;
};

```

ClickHouse尝试找到一个名为 `x.y.z` (或 `x_y_z` 或 `X.y_Z` 等)。

嵌套消息适用于输入或输出一个 **嵌套数据结构**。

在protobuf模式中定义的默认值，如下所示

```

syntax = "proto2";

message MessageType {
    optional int32 result_per_page = 3 [default = 10];
}

```

不应用;该 **表默认值** 用来代替它们。

ClickHouse在输入和输出protobuf消息 `length-delimited` 格式。

这意味着每个消息之前，应该写它的长度作为一个 `varint`。

另请参阅 [如何在流行语言中读取/写入长度分隔的protobuf消息](#).

Avro

[Apache Avro](#) 是在Apache Hadoop项目中开发的面向行的数据序列化框架。

ClickHouse Avro格式支持读取和写入 [Avro数据文件](#).

数据类型匹配 {#sql_reference/data_types-matching}

下表显示了支持的数据类型以及它们如何匹配ClickHouse **数据类型** 在 `INSERT` 和 `SELECT` 查询。

Avro数据类型 <code>INSERT</code>	ClickHouse数据类型	Avro数据类型 <code>SELECT</code>
<code>boolean, int, long, float, double</code>	<code>Int(8/16/32), UInt(8/16/32)</code>	<code>int</code>
<code>boolean, int, long, float, double</code>	<code>Int64, UInt64</code>	<code>long</code>
<code>boolean, int, long, float, double</code>	<code>Float32</code>	<code>float</code>

Zoci-avro数据类型 <code>INSERT</code>	ClickHouse数据类型	ClickHouse数据类型 <code>SELECT</code>
<code>bytes</code> , <code>string</code> , <code>fixed</code> , <code>enum</code>	字符串	<code>bytes</code>
<code>bytes</code> , <code>string</code> , <code>fixed</code>	固定字符串(N)	<code>fixed(N)</code>
<code>enum</code>	枚举(8/16)	<code>enum</code>
<code>array(T)</code>	阵列(T)	<code>array(T)</code>
<code>union(null, T)</code> , <code>union(T, null)</code>	可为空(T)	<code>union(null, T)</code>
<code>null</code>	可为空 (无)	<code>null</code>
<code>int (date) *</code>	日期	<code>int (date) *</code>
<code>long (timestamp-millis) *</code>	<code>DateTime64(3)</code>	<code>long (timestamp-millis) *</code>
<code>long (timestamp-micros) *</code>	<code>DateTime64(6)</code>	<code>long (timestamp-micros) *</code>

* Avro逻辑类型

不支持的Avro数据类型: `record` (非根), `map`

不支持的Avro逻辑数据类型: `uuid`, `time-millis`, `time-micros`, `duration`

插入数据

将Avro文件中的数据插入ClickHouse表:

```
$ cat file.avro | clickhouse-client --query="INSERT INTO {some_table} FORMAT Avro"
```

输入Avro文件的根模式必须是 `record` 类型。

要查找Avro schema的表列和字段之间的对应关系，ClickHouse比较它们的名称。此比较区分大小写。
跳过未使用的字段。

ClickHouse表列的数据类型可能与插入的Avro数据的相应字段不同。插入数据时，ClickHouse根据上表解释数据类型，然后 `投` 将数据转换为相应的列类型。

选择数据

从ClickHouse表中选择数据到Avro文件:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Avro" > file.avro
```

列名必须:

- 名,名,名,名 `[A-Za-z_]`
- 随后只包含 `[A-Za-z0-9_]`

输出Avro文件压缩和同步间隔可以配置 `output_format_avro_codec` 和 `output_format_avro_sync_interval` 分别。

AvroConfluent

AvroConfluent支持解码单对象Avro消息常用于 [卡夫卡](#) 和 [汇合的模式注册表](#).

每个Avro消息都嵌入了一个架构id，该架构id可以在架构注册表的帮助下解析为实际架构。

模式解析后会进行缓存。

架构注册表URL配置为 `format_avro_schema_registry_url`

数据类型匹配{#sql_reference/data_types-matching-1}

和 `Avro`

用途

要快速验证架构解析，您可以使用 `kafkacat` 与 `clickhouse-local`：

```
$ kafkacat -b kafka-broker -C -t topic1 -o beginning -f '%s' -c 3 | clickhouse-local --input-format AvroConfluent --format_avro_schema_registry_url 'http://schema-registry' -S "field1 Int64, field2 String" -q 'select * from table'  
1 a  
2 b  
3 c
```

使用 `AvroConfluent` 与 `clickhouse-local`：

```
CREATE TABLE topic1_stream  
(  
    field1 String,  
    field2 String  
)  
ENGINE = Kafka()  
SETTINGS  
kafka_broker_list = 'kafka-broker',  
kafka_topic_list = 'topic1',  
kafka_group_name = 'group1',  
kafka_format = 'AvroConfluent';  
  
SET format_avro_schema_registry_url = 'http://schema-registry';  
  
SELECT * FROM topic1_stream;
```

警告

设置 `format_avro_schema_registry_url` 需要在配置 `users.xml` `restart` 动后保持它的价值。

阿帕奇HDFS

阿帕奇HDFS 是Hadoop生态系统中普遍存在的列式存储格式。 ClickHouse支持此格式的读写操作。

数据类型匹配{#sql_reference/data_types-matching-2}

下表显示了支持的数据类型以及它们如何匹配ClickHouse `数据类型` 在 `INSERT` 和 `SELECT` 查询。

Parquet数据类型 (<code>INSERT</code>)	ClickHouse数据类型	Parquet数据类型 (<code>SELECT</code>)
<code>UINT8</code> , <code>BOOL</code>	<code>UInt8</code>	<code>UINT8</code>
<code>INT8</code>	<code>Int8</code>	<code>INT8</code>
<code>UINT16</code>	<code>UInt16</code>	<code>UINT16</code>

Parquet数据类型 (INSERT)	ClickHouse数据类型	Parquet数据类型 (SELECT)
UINT32	UInt32	UINT32
INT32	Int32	INT32
UINT64	UInt64	UINT64
INT64	Int64	INT64
FLOAT, HALF_FLOAT	Float32	FLOAT
DOUBLE	Float64	DOUBLE
DATE32	日期	UINT16
DATE64, TIMESTAMP	日期时间	UINT32
STRING, BINARY	字符串	STRING
—	固定字符串	STRING
DECIMAL	十进制	DECIMAL

ClickHouse支持可配置的精度 `Decimal` 类型。该 `INSERT` 查询对待实木复合地板 `DECIMAL` 键入为 ClickHouse `Decimal128` 类型。

不支持的Parquet数据类型: `DATE32`, `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

ClickHouse表列的数据类型可能与插入的Parquet数据的相应字段不同。插入数据时, ClickHouse根据上表解释数据类型, 然后 投 为ClickHouse表列设置的数据类型的数据。

插入和选择数据

您可以通过以下命令将Parquet数据从文件插入到ClickHouse表中:

```
$ cat {filename} | clickhouse-client --query="INSERT INTO {some_table} FORMAT Parquet"
```

您可以从ClickHouse表中选择数据, 并通过以下命令将它们保存到Parquet格式的某个文件中:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Parquet" > {some_file.pq}
```

要与Hadoop交换数据, 您可以使用 [HDFS表引擎](#).

ORC

阿帕奇兽人 是Hadoop生态系统中普遍存在的列式存储格式。您只能将此格式的数据插入ClickHouse。

数据类型匹配 {#sql_reference/data_types-matching-3}

下表显示了支持的数据类型以及它们如何匹配ClickHouse 数据类型 在 `INSERT` 查询。

ORC数据类型 (INSERT)	ClickHouse数据类型
UINT8, BOOL	UInt8

ORC数据类型 (INSERT)	ClickHouse数据类型
INT8	Int8
UINT16	UInt16
INT16	Int16
UINT32	UInt32
INT32	Int32
UINT64	UInt64
INT64	Int64
FLOAT, HALF_FLOAT	Float32
DOUBLE	Float64
DATE32	日期
DATE64, TIMESTAMP	日期时间
STRING, BINARY	字符串
DECIMAL	十进制

ClickHouse支持的可配置精度 Decimal 类型。该 INSERT 查询对待兽人 DECIMAL 键入为 ClickHouse Decimal128 类型。

不支持的ORC数据类型: DATE32, TIME32, FIXED_SIZE_BINARY, JSON, UUID, ENUM.

ClickHouse表列的数据类型不必匹配相应的ORC数据字段。插入数据时, ClickHouse根据上表解释数据类型, 然后 投 将数据转换为ClickHouse表列的数据类型集。

插入数据

您可以通过以下命令将文件中的ORC数据插入到ClickHouse表中:

```
$ cat filename.orc | clickhouse-client --query="INSERT INTO some_table FORMAT ORC"
```

要与Hadoop交换数据, 您可以使用 [HDFS表引擎](#).

格式架构

包含格式架构的文件名由该设置设置 format_schema.

当使用其中一种格式时, 需要设置此设置 Cap'n Proto 和 Protobuf.

格式架构是文件名和此文件中消息类型的名称的组合, 用冒号分隔,

e.g. schemafile.proto:MessageType.

如果文件具有格式的标准扩展名 (例如, .proto 为 Protobuf),

它可以被省略, 在这种情况下, 格式模式如下所示 schemafile:MessageType.

如果您通过输入或输出数据 客户 在交互模式下, 格式架构中指定的文件名

可以包含绝对路径或相对于客户端上当前目录的路径。

如果在批处理模式下使用客户端, 则由于安全原因, 架构的路径必须是相对的。

如果你通过输入或输出数据 [HTTP接口](#) 格式架构中指定的文件名

应该位于指定的目录中 `format_schema_path`
在服务器配置中。

跳过错误

一些格式，如 `CSV`, `TabSeparated`, `TSKV`, `JSONEachRow`, `Template`, `CustomSeparated` 和 `Protobuf` 如果发生解析错误，可以跳过断开的行，并从下一行开始继续解析。看 `input_format_allow_errors_num` 和 `input_format_allow_errors_ratio` 设置。

限制：

- 在解析错误的情况下 `JSONEachRow` 跳过所有数据，直到新行（或EOF），所以行必须由 `\n` 正确计算错误。
- `Template` 和 `CustomSeparated` 在最后一列之后使用分隔符，并在行之间使用分隔符来查找下一行的开头，所以跳过错误只有在其中至少有一个不为空时才有效。

要求

CPU

对于从预构建的deb包进行安装，请使用具有x86_64架构并支持SSE4.2指令的CPU。要使用不支持SSE4.2或具有AArch64或PowerPC64LE体系结构的处理器运行ClickHouse，您应该从源代码构建ClickHouse。

ClickHouse实现并行数据处理并使用所有可用的硬件资源。在选择处理器时，考虑到ClickHouse在具有大量内核但时钟速率较低的配置中的工作效率要高于具有较少内核和较高时钟速率的配置。例如，具有2600MHz的16核心优于具有3600MHz的8核心。

建议使用 涡轮增压 和 超线程 技术。它显着提高了典型工作负载的性能。

RAM

我们建议使用至少4GB的RAM来执行非平凡的查询。ClickHouse服务器可以使用少得多的RAM运行，但它需要处理查询的内存。

RAM所需的体积取决于：

- 查询的复杂性。
- 在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`, `DISTINCT`, `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。看 [在外部存储器中分组](#) 有关详细信息。

交换文件

禁用生产环境的交换文件。

存储子系统

您需要有2GB的可用磁盘空间来安装ClickHouse。

数据所需的存储量应单独计算。评估应包括：

- 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。

- 的数据压缩系数。

要估计数据压缩系数，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。例如，点击流数据通常被压缩6-10次。

要从其存储的快照中恢复，通常必须应用到快照的快照。如果计划将快照存储在多个副本中，则将快照的卷数以副本数。

网络

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。

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。该 tzdata 软件包应安装在系统中。

ClickHouse也可以在其他操作系统系列中工作。查看详细信息 [开始](#) 文档的部分。

疑难解答

- [安装方式](#)
- [连接到服务器](#)
- [查询处理](#)
- [查询处理效率](#)

安装方式

您无法使用Apt-get从ClickHouse存储库获取Deb软件包

- 检查防火墙设置。
- 如果出于任何原因无法访问存储库，请按照以下文件中的描述下载软件包 [开始](#) 文章并使用手动安装它们 `sudo dpkg -i <packages>` 指挥部 您还需要 tzdata 包。

连接到服务器

可能出现的问题：

- 服务器未运行。
- 意外或错误的配置参数。

服务器未运行

检查服务器是否运行

命令：

```
$ sudo service clickhouse-server status
```

如果服务器没有运行，请使用以下命令启动它：

```
$ sudo service clickhouse-server start
```

检查日志

主日志 clickhouse-server 是在 `/var/log/clickhouse-server/clickhouse-server.log` 默认情况下。

如果服务器成功启动，您应该看到字符串：

- `<Information> Application: starting up. — Server started.`
- `<Information> Application: Ready for connections. — Server is running and ready for connections.`

如果 clickhouse-server 启动失败与配置错误，你应该看到 `<Error>` 具有错误描述的字符串。例如：

```
2019.01.11 15:23:25.549505 [ 45 ] {} <Error> ExternalDictionaries: Failed reloading 'event2id' external dictionary: Poco::Exception. Code: 1000, e.code() = 111, e.displayText() = Connection refused, e.what() = Connection refused
```

如果在文件末尾没有看到错误，请从字符串开始查看整个文件：

```
<Information> Application: starting up.
```

如果您尝试启动第二个实例 `clickhouse-server` 在服务器上，您将看到以下日志：

```
2019.01.11 15:25:11.151730 [ 1 ] {} <Information> : Starting ClickHouse 19.1.0 with revision 54413
2019.01.11 15:25:11.154578 [ 1 ] {} <Information> Application: starting up
2019.01.11 15:25:11.156361 [ 1 ] {} <Information> StatusFile: Status file ./status already exists - unclean restart.
Contents:
PID: 8510
Started at: 2019-01-11 15:24:23
Revision: 54413

2019.01.11 15:25:11.156673 [ 1 ] {} <Error> Application: DB::Exception: Cannot lock file ./status. Another server instance in same directory is already running.
2019.01.11 15:25:11.156682 [ 1 ] {} <Information> Application: shutting down
2019.01.11 15:25:11.156686 [ 1 ] {} <Debug> Application: Uninitializing subsystem: Logging Subsystem
2019.01.11 15:25:11.156716 [ 2 ] {} <Information> BaseDaemon: Stop SignalListener thread
```

请参阅系统。**d** 日志

如果你没有找到任何有用的信息 `clickhouse-server` 日志或没有任何日志，您可以查看 `system.d` 使用命令记录：

```
$ sudo journalctl -u clickhouse-server
```

在交互模式下启动**clickhouse**服务器

```
$ sudo -u clickhouse /usr/bin/clickhouse-server --config-file /etc/clickhouse-server/config.xml
```

此命令将服务器作为带有自动启动脚本标准参数的交互式应用程序启动。在这种模式下 `clickhouse-server` 打印控制台中的所有事件消息。

配置参数

检查：

- 码头工人设置。

如果您在IPv6网络中的Docker中运行ClickHouse，请确保 `network=host` 已设置。

- 端点设置。

检查 `listen_host` 和 `tcp_port` 设置。

ClickHouse服务器默认情况下仅接受本地主机连接。

- HTTP协议设置。

检查HTTP API的协议设置。

- 安全连接设置。

检查：

- 该 `tcp_port_secure` 设置。
- 设置 `SSL`序列。

连接时使用正确的参数。例如，使用 `port_secure` 参数 `clickhouse_client`.

- 用户设置。

您可能使用了错误的用户名或密码。

查询处理

如果ClickHouse无法处理查询，它会向客户端发送错误描述。在 `clickhouse-client` 您可以在控制台中获得错误的描述。如果您使用的是HTTP接口，ClickHouse会在响应正文中发送错误描述。例如：

```
$ curl 'http://localhost:8123/' --data-binary "SELECT a"  
Code: 47, e.displayText() = DB::Exception: Unknown identifier: a. Note that there are no tables (FROM clause) in your  
query, context: required_names: 'a' source_tables: table_aliases: private_aliases: column_aliases: public_columns: 'a'  
masked_columns: array_join_columns: source_columns: , e.what() = DB::Exception
```

如果你开始 `clickhouse-client` 与 `stack-trace` 参数，ClickHouse返回包含错误描述的服务器堆栈跟踪。

您可能会看到一条关于连接中断的消息。在这种情况下，可以重复查询。如果每次执行查询时连接中断，请检查服务器日志中是否存在错误。

查询处理效率

如果您发现ClickHouse工作速度太慢，则需要为查询分析服务器资源和网络的负载。

您可以使用`clickhouse-benchmark`实用程序来分析查询。它显示每秒处理的查询数、每秒处理的行数以及查询处理时间的百分位数。

点击更新

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
```

如果您使用除推荐的deb包之外的其他内容安装ClickHouse，请使用适当的更新方法。

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

碌莽禄While: `复制` provides protection from hardware failures, it does not protect against human errors: accidental deletion of data, deletion of the wrong table or a table on the wrong cluster, and software bugs that result in incorrect data processing or data corruption. In many cases mistakes like these will affect all replicas. ClickHouse has built-in safeguards to prevent some types of mistakes — for example, by default 您不能使用类似MergeTree的引擎删除包含超过50Gb数据的表. 但是，这些保障措施并不涵盖所有可能的情况，可以规避。

为了有效地减少可能的人为错误，您应该仔细准备备份和还原数据的策略 提前.

每家公司都有不同的可用资源和业务需求，因此没有适合各种情况的ClickHouse备份和恢复通用解决方案。什么适用于一千兆字节的数据可能不会为几十pb的工作。有多种可能的方法有自己的优点和缺点，这将在下面讨论。这是一个好主意，使用几种方法，而不是只是一个，以弥补其各种缺点。

注

请记住，如果您备份了某些内容并且从未尝试过还原它，那么当您实际需要它时（或者至少需要比业务能够容忍的时间更长），恢复可能无法正常工作。因此，无论您选择哪种备份方法，请确保自动还原过程，并定期在备用ClickHouse群集上练习。

将源数据复制到其他地方

通常被摄入到ClickHouse的数据是通过某种持久队列传递的，例如 [Apache Kafka](#). 在这种情况下，可以配置一组额外的订阅服务器，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的推荐冷存储，可能是对象存储或分布式文件系统，如 [HDFS](#).

文件系统快照

某些本地文件系统提供快照功能（例如，[ZFS](#)），但它们可能不是提供实时查询的最佳选择。一个可能的解决方案是使用这种文件系统创建额外的副本，并将它们从 [分布](#) 用于以下目的的表 `SELECT` 查询。任何修改数据的查询都无法访问此类副本上的快照。作为奖励，这些副本可能具有特殊的硬件配置，每个服务器附加更多的磁盘，这将是经济高效的。

ツ環板-ヨツ嘉ツツ偲

ツ環板-ヨツ嘉ツツ偲 是一个多功能工具，最初创建用于重新分片 pb 大小的表。它还可用于备份和还原目的，因为它可以在 ClickHouse 表和集群之间可靠地复制数据。

对于较小的数据量，一个简单的 `INSERT INTO ... SELECT ...` 到远程表也可以工作。

部件操作

ClickHouse 允许使用 `ALTER TABLE ... FREEZE PARTITION ...` 查询以创建表分区的本地副本。这是使用硬链接来实现 `/var/lib/clickhouse/shadow/` 文件夹中，所以它通常不会占用旧数据的额外磁盘空间。创建的文件副本不由 ClickHouse 服务器处理，所以你可以把它们留在那里：你将有一个简单的备份，不需要任何额外的外部系统，但它仍然会容易出现硬件问题。出于这个原因，最好将它们远程复制到另一个位置，然后删除本地副本。分布式文件系统和对象存储仍然是一个不错的选择，但是具有足够大容量的正常附加文件服务器也可以工作（在这种情况下，传输将通过网络文件系统 [rsync](#)）。

有关与分区操作相关的查询的详细信息，请参阅 [更改文档](#)。

第三方工具可用于自动化此方法：[ツ環板backupヨツ嘉ツツ偲](#)。

采样查询探查器

ClickHouse 运行允许分析查询执行的采样探查器。使用探查器，您可以找到在查询执行期间使用最频繁的源代码例程。您可以跟踪 CPU 时间和挂钟花费的时间，包括空闲时间。

使用概要分析器：

- 设置 [trace_log](#) 服务器配置部分。

本节配置 [trace_log](#) 系统表包含探查器运行的结果。它是默认配置的。请记住，此表中的数据仅对正在运行的服务器有效。服务器重新启动后，ClickHouse 不会清理表，所有存储的虚拟内存地址都可能无效。

- 设置 [query_profiler_cpu_time_period_ns](#) 或 [query_profiler_real_time_period_ns](#) 设置。这两种设置可以同时使用。

这些设置允许您配置探查器计时器。由于这些是会话设置，您可以为整个服务器、单个用户或用户配置文件、交互式会话以及每个单个查询获取不同的采样频率。

默认采样频率为每秒一个采样，CPU和实时定时器都启用。该频率允许收集有关ClickHouse集群的足够信息。同时，使用此频率，profiler不会影响ClickHouse服务器的性能。如果您需要分析每个单独的查询，请尝试使用更高的采样频率。

分析 `trace_log` 系统表：

- 安装 `clickhouse-common-static-dbg` 包。看 [从DEB软件包安装](#)。
- 允许由内省功能 `allow_introspection_functions` 设置。

出于安全原因，默认情况下禁用内省功能。

- 使用 `addressToLine`, `addressToSymbol` 和 `demangle` 内省功能 获取函数名称及其在ClickHouse代码中的位置。要获取某些查询的配置文件，您需要从以下内容汇总数据 `trace_log` 桌子 您可以通过单个函数或整个堆栈跟踪聚合数据。

如果你需要想象 `trace_log` 信息，尝试 [flamegraph](#) 和 [测速镜](#)。

示例

在这个例子中，我们：

- 过滤 `trace_log` 数据由查询标识符和当前日期组成。
- 通过堆栈跟踪聚合。
- 使用内省功能，我们将得到一个报告：
 - 符号名称和相应的源代码函数。
 - 这些函数的源代码位置。

```
SELECT
    count(),
    arrayStringConcat(arrayMap(x -> concat(demangle(addressToSymbol(x)), '\n  ', addressToLine(x)), trace), '\n') AS sym
FROM system.trace_log
WHERE (query_id = 'ebca3574-ad0a-400a-9cbc-dca382f5998c') AND (event_date = today())
GROUP BY trace
ORDER BY count() DESC
LIMIT 10
```

Row 1:

```
count(): 6344
sym: StackTrace::StackTrace(ucontext_t const&)
  /home/milovidov/ClickHouse/build_gcc9/..../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, signinfo_t*, void*) [clone .isra.0]
  /home/milovidov/ClickHouse/build_gcc9/..../src/IO/BufferBase.h:99
```

read

```
DB::ReadBufferFromFileDescriptor::nextImpl()
  /home/milovidov/ClickHouse/build_gcc9/..../src/IO/ReadBufferFromFileDescriptor.cpp:56
DB::CompressedReadBufferBase::readCompressedData(unsigned long&, unsigned long&)
  /home/milovidov/ClickHouse/build_gcc9/..../src/IO/ReadBuffer.h:54
DB::CompressedReadBufferFromFile::nextImpl()
  /home/milovidov/ClickHouse/build_gcc9/..../src/Compression/CompressedReadBufferFromFile.cpp:22
DB::CompressedReadBufferFromFile::seek(unsigned long, unsigned long)
  /home/milovidov/ClickHouse/build_gcc9/..../src/Compression/CompressedReadBufferFromFile.cpp:63
DB::MergeTreeReaderStream::seekToMark(unsigned long)
  /home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReaderStream.cpp:200
std::_Function_handler<DB::ReadBuffer* (std::vector<DB::IDataType::Substream,
```

```

std::allocator<DB::IDataType::Substream> > const&),
DB::MergeTreeReader::readData(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool):::{lambda(bool)#1}::operator()
(bool) const:{lambda(std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> >
const&)#1}>::_M_invoke(std::Any_data const&, std::vector<DB::IDataType::Substream,
std::allocator<DB::IDataType::Substream> > const&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeReader.cpp:212
DB::IDataType::deserializeBinaryBulkWithMultipleStreams(DB::IColumn&, unsigned long,
DB::IDataType::DeserializeBinaryBulkSettings&, std::shared_ptr<DB::IDataType::DeserializeBinaryBulkState>&) const
/usr/local/include/c++/9.1.0/bits/std_function.h:690
DB::MergeTreeReader::readData(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeReader.cpp:232
DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeReader.cpp:111
DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeRangeReader.cpp:35
DB::MergeTreeRangeReader::continueReadingChain(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeRangeReader.cpp:219
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeRangeReader.cpp:487
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*>,
std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*__&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&,
std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&):{lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

Row 2:

count(): 3295
sym: StackTrace::StackTrace(ucontext_t const&)

```

/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99

__pthread_cond_wait

std::condition_variable::wait(std::unique_lock<std::mutex>&)
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/src/c++11/../../../../gcc-
9.1.0/libstdc++-v3/src/c++11/condition_variable.cc:55
Poco::Semaphore::wait()
/home/milovidov/ClickHouse/build_gcc9/../contrib/poco/Foundation/src/Semaphore.cpp:61
DB::UnionBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/x86_64-pc-linux-gnu/bits/gthr-default.h:748
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/Core/Block.h:90
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::LimitBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::AsynchronousBlockInputStream::calculate()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
std::_Function_handler<void (), DB::AsynchronousBlockInputStream::next()::{lambda()#1}>::__M_invoke(std::__Any_data
const&)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:551
ThreadPoolImpl<ThreadFromGlobalPool>::worker(std::__List_iterator<ThreadFromGlobalPool>)
/usr/local/include/c++/9.1.0/x86_64-pc-linux-gnu/bits/gthr-default.h:748
ThreadFromGlobalPool::ThreadFromGlobalPool<ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>
(std::function<void ()>, int, std::optional<unsigned long>::{lambda()#3}>
(ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>(std::function<void ()>, int, std::optional<unsigned
long>::{lambda()#3} &&::{lambda()#1}::operator()() const
/home/milovidov/ClickHouse/build_gcc9/../src/Common/ThreadPool.h:146
ThreadPoolImpl<std::thread>::worker(std::__List_iterator<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

Row 3:

```

count(): 1978
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99

```

```

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true>>::search(unsigned char const*, unsigned long) const
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::MatchImpl<true, false>::vector_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false,
AllocatorHint::DefaultHint 67108864ul, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul>

```

```

AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&,
std::vector<unsigned long, std::allocator<unsigned long>> const&, unsigned long, unsigned long)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long>> const&, unsigned long, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/..../src/Functions/IFunction.cpp:464
DB::ExpressionAction::execute(DB::Block&, bool) const
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::ExpressionActions::execute(DB::Block&, bool) const
/home/milovidov/ClickHouse/build_gcc9/..../src/Interpreters/ExpressionActions.cpp:739
DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange>>&
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange>>&
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&){lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::_List_iterator<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/include/bits/unique_ptr.h:81
start_thread

__clone
```

Row 4:

```

count(): 1913
sym: StackTrace::StackTrace(ucontext_t const&
/home/milovidov/ClickHouse/build_gcc9/..../src/Common/StackTrace.cpp:208
```

```
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/I/O/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true>>::search(unsigned char const*, unsigned long) const
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::MatchImpl<true, false>::vector_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long>> const&, unsigned long, unsigned long)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long>> const&, unsigned long, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/../src/Functions/IFunction.cpp:464
DB::ExpressionAction::execute(DB::Block&, bool) const
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::ExpressionActions::execute(DB::Block&, bool) const
/home/milovidov/ClickHouse/build_gcc9/../src/Interpreters/ExpressionActions.cpp:739
DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange>>&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange>>&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&){lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/include/bits/unique_ptr.h:81
start_thread
```

_clone

Row 5:

```
count(): 1672
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const*, unsigned long) const
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::MatchImpl<true, false>::vector_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false,
AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&,
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char,
4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&,
std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&,
unsigned long, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/../src/Functions/IFunction.cpp:464
DB::ExpressionAction::execute(DB::Block&, bool) const
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::ExpressionActions::execute(DB::Block&, bool) const
/home/milovidov/ClickHouse/build_gcc9/../src/Interpreters/ExpressionActions.cpp:739
DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*>,
std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&,
std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&){lambda()#1}::operator()() const
```

```
/usr/local/include/c++/9.1.0/bits/snared_ptr_base.h:29
ThreadPoolImpl<std::thread>::worker(std::_List_iterator<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread
```

_clone

Row 6:

```
count(): 1531
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/..../src/IO/BufferBase.h:99
```

read

```
DB::ReadBufferFromFileDescriptor::nextImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/IO/ReadBufferFromFileDescriptor.cpp:56
DB::CompressedReadBufferBase::readCompressedData(unsigned long&, unsigned long&)
/home/milovidov/ClickHouse/build_gcc9/..../src/IO/ReadBuffer.h:54
DB::CompressedReadBufferFromFile::nextImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/Compression/CompressedReadBufferFromFile.cpp:22
void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false,
AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..../src/IO/ReadBuffer.h:53
DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const
/home/milovidov/ClickHouse/build_gcc9/..../src/DataTypes/DataTypeString.cpp:202
DB::MergeTreeReader::readData(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:232
DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:111
DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35
DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange,
std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
```

```
DB::ParallelInputsProcessor::execute(DB::UnionBlockInputStream::Handler> <lambda>)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*>,
std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&,
std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&):{lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone
```

Row 7:

```
count(): 1034
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/..src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/..src/IO/BufferBase.h:99
```

```
DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const*, unsigned long) const
/opt/milovidov/ClickHouse/build_gcc9/programs	clickhouse
DB::MatchImpl<true, false>::vector_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false>,
AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&,
std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char,
4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)
/opt/milovidov/ClickHouse/build_gcc9/programs	clickhouse
DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&,
std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)
/opt/milovidov/ClickHouse/build_gcc9/programs	clickhouse
DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&,
unsigned long, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/..src/Functions/IFunction.cpp:464
DB::ExpressionAction::execute(DB::Block&, bool) const
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::ExpressionActions::execute(DB::Block&, bool) const
/home/milovidov/ClickHouse/build_gcc9/..src/Interpreters/ExpressionActions.cpp:739
DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeRangeReader.cpp:660
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeRangeReader.cpp:546
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
```

```
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&)>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&){lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::_List_iterator<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/include/bits/unique_ptr.h:81
start_thread
```

__clone

Row 8:

```
count(): 989
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99
```

__lli_lock_wait

pthread_mutex_lock

```
DB::MergeTreeReaderStream::loadMarks()
/usr/local/include/c++/9.1.0/bits/std_mutex.h:103
DB::MergeTreeReaderStream::MergeTreeReaderStream(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> > const&, DB::MarkCache*, bool, DB::UncompressedCache*, unsigned long, unsigned long, unsigned long, DB::MergeTreeIndexGranularityInfo const*, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeReaderStream.cpp:107
std::function_handler<void (std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&), DB::MergeTreeReader::addStreams(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, DB::IDataType const&, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int){lambda(std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&#1}>::_M_invoke(std::Any_data const&, std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)
/usr/local/include/c++/9.1.0/bits/unique_ptr.h:147
DB::MergeTreeReader::addStreams(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, DB::IDataType const&, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::MergeTreeReader::MergeTreeReader(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
```

```

> const&, std::shared_ptr<DB::MergeTreeDataPart const> const&, DB::NamesAndTypesList const&,
DB::UncompressedCache*, DB::MarkCache*, bool, DB::MergeTreeData const&, std::vector<DB::MarkRange,
std::allocator<DB::MarkRange> > const&, unsigned long, unsigned long, std::map<std::basic_string<char,
std::char_traits<char>, std::allocator<char> >, double, std::less<std::basic_string<char,
std::char_traits<char>, std::allocator<char> >, std::allocator<std::pair<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > const, double> >> const&, std::function<void
(DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)
    /usr/local/include/c++/9.1.0/bits/stl_list.h:303
DB::MergeTreeThreadSelectBlockInputStream::getNewTask()
    /usr/local/include/c++/9.1.0/bits/std_function.h:259
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
    /home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:54
DB::IBlockInputStream::read()
    /usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
    /home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
    /usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
    /home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
    /usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
    /usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
    /home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*>,
std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&)(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&,
std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const
    /usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
    /usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
    /home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

Row 9:

```

count(): 779
sym: StackTrace::StackTrace(ucontext_t const&)
    /home/milovidov/ClickHouse/build_gcc9/..src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
    /home/milovidov/ClickHouse/build_gcc9/..src/IO/BufferBase.h:99

```

```

void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false,
AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)
    /usr/local/lib/gcc/x86_64-pc-linux-gnu/9.1.0/include/emmintrin.h:727
DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const
    /home/milovidov/ClickHouse/build_gcc9/..src/DataTypes/DataTypeString.cpp:202
DB::MergeTreeReader::readData(std::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)
    /home/milovidov/ClickHouse/build_gcc9/..src/Storages/MergeTree/MergeTreeReader.cpp:232

```

```

/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:252
DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:111
DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35
DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange,
std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*>,
std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*__&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&,
std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&):{lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolsImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

Row 10:

```

count(): 666
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/..../src/IO/BufferBase.h:99

```

```

void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false,
AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul,
AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)
/usr/local/lib/gcc/x86_64-pc-linux-gnu/9.1.0/include/emmintrin.h:727
DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const

```

```

/home/milovidov/ClickHouse/build_gcc9/..../src/DataTypes/DataTypeString.cpp:202
DB::MergeTreeReader::readData(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:232
DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeReader.cpp:111
DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35
DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange,
std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long)
/home/milovidov/ClickHouse/build_gcc9/..../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&>(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)(std::shared_ptr<DB::ThreadGroupStatus>,
unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&):{lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::_List_iterator<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

系统表

系统表用于实现系统的部分功能，并提供对有关系统如何工作的信息的访问。

您无法删除系统表（但可以执行分离）。

系统表没有包含磁盘上数据的文件或包含元数据的文件。服务器在启动时创建所有系统表。

系统表是只读的。

它们位于 ‘system’ 数据库。

系统。asynchronous_metrics

包含在后台定期计算的指标。例如，在使用的RAM量。

列:

- `metric` (字符串) — Metric name.
- `value` (Float64) — Metric value.

示例

```
SELECT * FROM system.asynchronous_metrics LIMIT 10
```

metric	value
jemalloc.background_thread.run_interval	0
jemalloc.background_thread.num_runs	0
jemalloc.background_thread.num_threads	0
jemalloc.retained	422551552
jemalloc.mapped	1682989056
jemalloc.resident	1656446976
jemalloc.metadata_thp	0
jemalloc.metadata	10226856
UncompressedCacheCells	0
MarkCacheFiles	0

另请参阅

- [监测](#) — Base concepts of ClickHouse monitoring.
- [系统。指标](#) — Contains instantly calculated metrics.
- [系统。活动](#) — Contains a number of events that have occurred.
- [系统。metric_log](#) — Contains a history of metrics values from tables `system.metrics` и `system.events`.

系统。集群

包含有关配置文件中可用的集群及其中的服务器的信息。

列:

- `cluster` (String) — The cluster name.
- `shard_num` (UInt32) — The shard number in the cluster, starting from 1.
- `shard_weight` (UInt32) — The relative weight of the shard when writing data.
- `replica_num` (UInt32) — The replica number in the shard, starting from 1.
- `host_name` (String) — The host name, as specified in the config.
- `host_address` (String) — The host IP address obtained from DNS.
- `port` (UInt16) — The port to use for connecting to the server.
- `user` (String) — The name of the user for connecting to the server.
- `errors_count` (UInt32)-此主机无法到达副本的次数。
- `estimated_recovery_time` (UInt32)-剩下的秒数，直到副本错误计数归零，它被认为是恢复正常。

请注意 `errors_count` 每个查询集群更新一次，但 `estimated_recovery_time` 按需重新计算。所以有可能是非零的情况 `errors_count` 和零 `estimated_recovery_time`，下一个查询将为零 `errors_count` 并尝试使用副本，就好像它没有错误。

另请参阅

- [表引擎分布式](#)
- [distributed_replica_error_cap](#)设置
- [distributed_replica_error_half_life](#)设置

系统。列

包含有关所有表中列的信息。

您可以使用此表获取类似于以下内容的信息 **DESCRIBE TABLE** 查询，但对于多个表一次。

该 `system.columns` 表包含以下列（列类型显示在括号中）：

- `database` (String) — Database name.
- `table` (String) — Table name.
- `name` (String) — Column name.
- `type` (String) — Column type.
- `default_kind` (String) — Expression type (`DEFAULT`, `MATERIALIZED`, `ALIAS`) 为默认值，如果没有定义，则为空字符串。
- `default_expression` (String) — Expression for the default value, or an empty string if it is not defined.
- `data_compressed_bytes` (UInt64) — The size of compressed data, in bytes.
- `data_uncompressed_bytes` (UInt64) — The size of decompressed data, in bytes.
- `marks_bytes` (UInt64) — The size of marks, in bytes.
- `comment` (String) — Comment on the column, or an empty string if it is not defined.
- `is_in_partition_key` (UInt8) — Flag that indicates whether the column is in the partition expression.
- `is_in_sorting_key` (UInt8) — Flag that indicates whether the column is in the sorting key expression.
- `is_in_primary_key` (UInt8) — Flag that indicates whether the column is in the primary key expression.
- `is_in_sampling_key` (UInt8) — Flag that indicates whether the column is in the sampling key expression.

系统。贡献者

包含有关贡献者的信息。按随机顺序排列所有构造。该顺序在查询执行时是随机的。

列：

- `name` (String) — Contributor (author) name from git log.

示例

```
SELECT * FROM system.contributors LIMIT 10
```

name
Olga Khvostikova
Max Vetrov
LiuYangkuan
svladykin
zamulla
Šimon Podlipský
BayoNet
Ilya Khomutov
Amy Krishnevsky
Loud_Scream

要在表中找出自己，请使用查询：

```
SELECT * FROM system.contributors WHERE name='Olga Khvostikova'
```

name
Olga Khvostikova

系统。数据库

此表包含一个名为"字符串"的列 'name' – the name of a database.

服务器知道的每个数据库在表中都有相应的条目。

该系统表用于实现 SHOW DATABASES 查询。

系统。detached_parts

包含有关分离部分的信息 MergeTree 桌子 该 reason 列指定分离部件的原因。对于用户分离的部件，原因是空的。这些部件可以附加 ALTER TABLE ATTACH PARTITION|PART 指挥部 有关其他列的说明，请参阅 系统。零件。如果部件名称无效，某些列的值可能为 NULL。这些部分可以删除 ALTER TABLE DROP DETACHED PART.

系统。字典

包含以下信息 外部字典.

列:

- database (字符串) — Name of the database containing the dictionary created by DDL query. Empty string for other dictionaries.
- name (字符串) — 字典名称.
- status (枚举8) — Dictionary status. Possible values:
 - NOT_LOADED — Dictionary was not loaded because it was not used.
 - LOADED — Dictionary loaded successfully.
 - FAILED — Unable to load the dictionary as a result of an error.
 - LOADING — Dictionary is loading now.
 - LOADED_AND_RELOADING — Dictionary is loaded successfully, and is being reloaded right now (frequent reasons: SYSTEM RELOAD DICTIONARY 查询, 超时, 字典配置已更改)。
 - FAILED_AND_RELOADING — Could not load the dictionary as a result of an error and is loading now.
- origin (字符串) — Path to the configuration file that describes the dictionary.
- type (字符串) — Type of a dictionary allocation. 在内存中存储字典.
- key — 密钥类型: 数字键 (UInt64) or Composite key (字符串) — form "(type 1, type 2, ..., type n)".
- attribute.names (阵列(字符串)) — Array of 属性名称 由字典提供。
- attribute.types (阵列(字符串)) — Corresponding array of 属性类型 这是由字典提供。
- bytes_allocated (UInt64) — Amount of RAM allocated for the dictionary.
- query_count (UInt64) — Number of queries since the dictionary was loaded or since the last successful reboot.
- hit_rate (Float64) — For cache dictionaries, the percentage of uses for which the value was in the cache.
- element_count (UInt64) — Number of items stored in the dictionary.
- load_factor (Float64) — Percentage filled in the dictionary (for a hashed dictionary, the percentage filled in the hash table).
- source (字符串) — Text describing the 数据源 为了字典
- lifetime_min (UInt64) — Minimum 使用寿命 在内存中的字典，之后 ClickHouse 尝试重新加载字典 (如果 invalidate_query 被设置，那么只有当它已经改变)。在几秒钟内设置。
- lifetime_max (UInt64) — Maximum 使用寿命 在内存中的字典，之后 ClickHouse 尝试重新加载字典 (如果 invalidate_query 被设置，那么只有当它已经改变)。在几秒钟内设置。
- loading_start_time (日期时间) — Start time for loading the dictionary.
- last_successful_update_time (日期时间) — End time for loading or updating the dictionary. Helps to monitor some troubles with external sources and investigate causes.
- loading_duration (Float32) — Duration of a dictionary loading.
- last_exception (字符串) — Text of the error that occurs when creating or reloading the dictionary if the dictionary couldn't be created.

示例

和黑宝典。

```

CREATE DICTIONARY dictdb.dict
(
    `key` Int64 DEFAULT -1,
    `value_default` String DEFAULT 'world',
    `value_expression` String DEFAULT 'xxx' EXPRESSION 'toString(127 * 172)'
)
PRIMARY KEY key
SOURCE(CLICKHOUSE(HOST 'localhost' PORT 9000 USER 'default' TABLE 'dicttbl' DB 'dictdb'))
LIFETIME(MIN 0 MAX 1)
LAYOUT(FLAT())

```

确保字典已加载。

```
SELECT * FROM system.dictionaries
```

database	name	status	origin	type	key	attribute.names	attribute.types	bytes_allocated	query_count	hit_rate	element_count	load_factor	source	lifetime_min	lifetime_max	loading_start_time	last_successful_update_time	loading_duration	last_exception
dictdb	dict	LOADED	dictdb.dict	Flat	UInt64	['value_default','value_expression']	['String','String']	74032	0	1	1	0.0004887585532746823	ClickHouse: dictdb.dicttbl	0	1	2020-03-04 04:17:34	2020-03-04 04:30:34	0.002	

系统。活动

包含有关系统中发生的事件数的信息。例如，在表中，您可以找到多少 `SELECT` 自 ClickHouse 服务器启动以来已处理查询。

列：

- `event` (字符串) — Event name.
- `value` (UInt64) — Number of events occurred.
- `description` (字符串) — Event description.

示例

```
SELECT * FROM system.events LIMIT 5
```

event	value	description
Query	12	Number of queries to be interpreted and potentially executed. Does not include queries that failed to parse or were rejected due to AST size limits, quota limits or limits on the number of simultaneously running queries. May include internal queries initiated by ClickHouse itself. Does not count subqueries.
SelectQuery	8	Same as Query, but only for SELECT queries.
FileOpen	73	Number of files opened.
ReadBufferFromFileDescriptorRead	155	Number of reads (read/pread) from a file descriptor. Does not include sockets.

| ReadBufferFromFileDescriptorReadBytes | 9931 | Number of bytes read from file descriptors. If the file is compressed, this will show the compressed data size.

另请参阅

- 系统。asynchronous_metrics — Contains periodically calculated metrics.
- 系统。指标 — Contains instantly calculated metrics.
- 系统。metric_log — Contains a history of metrics values from tables system.metrics и system.events.
- 监测 — Base concepts of ClickHouse monitoring.

系统。功能

包含有关正常函数和聚合函数的信息。

列:

- name (String) – The name of the function.
- is_aggregate (UInt8) — Whether the function is aggregate.

系统。graphite_retentions

包含有关参数的信息 graphite_rollup 这是在表中使用 *GraphiteMergeTree 引擎

列:

- config_name (字符串) - graphite_rollup 参数名称。
- regexp (String)-指标名称的模式。
- function (String)-聚合函数的名称。
- age (UInt64)-以秒为单位的数据的最小期限。
- precision (UInt64) -如何精确地定义以秒为单位的数据的年龄。
- priority (UInt16)-模式优先级。
- is_default (UInt8)-模式是否为默认值。
- Tables.database (Array(String))-使用数据库表名称的数组 config_name 参数。
- Tables.table (Array(String))-使用表名称的数组 config_name 参数。

系统。合并

包含有关MergeTree系列中表当前正在进行的合并和部件突变的信息。

列:

- database (String) — The name of the database the table is in.
- table (String) — Table name.
- elapsed (Float64) — The time elapsed (in seconds) since the merge started.
- progress (Float64) — The percentage of completed work from 0 to 1.
- num_parts (UInt64) — The number of pieces to be merged.
- result_part_name (String) — The name of the part that will be formed as the result of merging.
- is_mutation (UInt8)-如果这个过程是一个部分突变。
- total_size_bytes_compressed (UInt64) — The total size of the compressed data in the merged chunks.
- total_size_marks (UInt64) — The total number of marks in the merged parts.
- bytes_read_uncompressed (UInt64) — Number of bytes read, uncompressed.
- rows_read (UInt64) — Number of rows read.
- bytes_written_uncompressed (UInt64) — Number of bytes written, uncompressed.
- rows_written (UInt64) — Number of rows written.

系统。指标

包含可以立即计算或具有当前值的指标。例如，同时处理的查询的数量或当前副本的延迟。此表始终是最新的。

列：

- `metric` (字符串) — Metric name.
- `value` (Int64) — Metric value.
- `description` (字符串) — Metric description.

支持的指标列表，您可以在 [src/Common/CurrentMetrics.cpp](#) ClickHouse的源文件。

示例

```
SELECT * FROM system.metrics LIMIT 10
```

metric	value	description
Query	1	Number of executing queries
Merge	0	Number of executing background merges
PartMutation	0	Number of mutations (ALTER DELETE/UPDATE)
ReplicatedFetch	0	Number of data parts being fetched from replicas
ReplicatedSend	0	Number of data parts being sent to replicas
ReplicatedChecks	0	Number of data parts checking for consistency
BackgroundPoolTask	0	Number of active tasks in BackgroundProcessingPool (merges, mutations, fetches, or replication queue bookkeeping)
BackgroundSchedulePoolTask	0	Number of active tasks in BackgroundSchedulePool. This pool is used for periodic ReplicatedMergeTree tasks, like cleaning old data parts, altering data parts, replica re-initialization, etc.
DiskSpaceReservedForMerge	0	Disk space reserved for currently running background merges. It is slightly more than the total size of currently merging parts.
DistributedSend	0	Number of connections to remote servers sending data that was INSERTed into Distributed tables. Both synchronous and asynchronous mode.

另请参阅

- [系统。asynchronous_metrics](#) — Contains periodically calculated metrics.
- [系统。活动](#) — Contains a number of events that occurred.
- [系统。metric_log](#) — Contains a history of metrics values from tables `system.metrics` и `system.events`.
- [监测](#) — Base concepts of ClickHouse monitoring.

系统。metric_log

包含表中度量值的历史记录 `system.metrics` 和 `system.events`，定期刷新到磁盘。

打开指标历史记录收集 `system.metric_log`, 创建 `/etc/clickhouse-server/config.d/metric_log.xml` 具有以下内容:

```
<yandex>
  <metric_log>
    <database>system</database>
```

```
<table>metric_log</table>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
<collect_interval_milliseconds>1000</collect_interval_milliseconds>
</metric_log>
</yandex>
```

示例

```
SELECT * FROM system.metric_log LIMIT 1 FORMAT Vertical;
```

Row 1:

event_date:	2020-02-18
event_time:	2020-02-18 07:15:33
milliseconds:	554
ProfileEvent_Query:	0
ProfileEvent_SelectQuery:	0
ProfileEvent_InsertQuery:	0
ProfileEvent_FileOpen:	0
ProfileEvent_Seek:	0
ProfileEvent_ReadBufferFromFileDescriptorRead:	1
ProfileEvent_ReadBufferFromFileDescriptorReadFailed:	0
ProfileEvent_ReadBufferFromFileDescriptorReadBytes:	0
ProfileEvent_WriteBufferFromFileDescriptorWrite:	1
ProfileEvent_WriteBufferFromFileDescriptorWriteFailed:	0
ProfileEvent_WriteBufferFromFileDescriptorWriteBytes:	56
...	
CurrentMetric_Query:	0
CurrentMetric_Merge:	0
CurrentMetric_PartMutation:	0
CurrentMetric_ReplicatedFetch:	0
CurrentMetric_ReplicatedSend:	0
CurrentMetric_ReplicatedChecks:	0
...	

另请参阅

- 系统。[asynchronous_metrics](#) — Contains periodically calculated metrics.
- 系统。[活动](#) — Contains a number of events that occurred.
- 系统。[指标](#) — Contains instantly calculated metrics.
- [监测](#) — Base concepts of ClickHouse monitoring.

系统。数字

此表包含一个名为 UInt64 的列 ‘number’ 它包含几乎所有从零开始的自然数。

您可以使用此表进行测试，或者如果您需要进行暴力搜索。

从此表中读取的内容不是并行的。

系统。numbers_mt

一样的 ‘system.numbers’ 但读取是并行的。 这些数字可以以任何顺序返回。
用于测试。

系统。一

此表包含一行，其中包含一行 ‘dummy’ UInt8 列包含值 0。
如果 SELECT 查询未指定 FROM 子句，则使用此表。
这与其他 Dbms 中的双表类似。

系统。零件

包含有关的部分信息 [MergeTree](#) 桌子

每行描述一个数据部分。

列:

- `partition` (`String`) – The partition name. To learn what a partition is, see the description of the [ALTER](#) 查询。

格式:

- `YYYYMM` 用于按月自动分区。
- `any_string` 手动分区时。

- `name` (`String`) – Name of the data part.

- `active` (`UInt8`) – Flag that indicates whether the data part is active. If a data part is active, it's used in a table. Otherwise, it's deleted. Inactive data parts remain after merging.

- `marks` (`UInt64`) – The number of marks. To get the approximate number of rows in a data part, multiply `marks` 通过索引粒度 (通常为 8192) (此提示不适用于自适应粒度)。

- `rows` (`UInt64`) – The number of rows.

- `bytes_on_disk` (`UInt64`) – Total size of all the data part files in bytes.

- `data_compressed_bytes` (`UInt64`) – Total size of compressed data in the data part. All the auxiliary files (for example, files with marks) are not included.

- `data_uncompressed_bytes` (`UInt64`) – Total size of uncompressed data in the data part. All the auxiliary files (for example, files with marks) are not included.

- `marks_bytes` (`UInt64`) – The size of the file with marks.

- `modification_time` (`DateTime`) – The time the directory with the data part was modified. This usually corresponds to the time of data part creation.|

- `remove_time` (`DateTime`) – The time when the data part became inactive.

- `refcount` (`UInt32`) – The number of places where the data part is used. A value greater than 2 indicates that the data part is used in queries or merges.

- `min_date` (`Date`) – The minimum value of the date key in the data part.

- `max_date` (`Date`) – The maximum value of the date key in the data part.

- `min_time` (`DateTime`) – The minimum value of the date and time key in the data part.

- `max_time` (`DateTime`) – The maximum value of the date and time key in the data part.

- `partition_id` (`String`) – ID of the partition.

- `min_block_number` (`UInt64`) – The minimum number of data parts that make up the current part after merging.

- `max_block_number` (`UInt64`) – The maximum number of data parts that make up the current part after merging.

- `level` (`UInt32`) – Depth of the merge tree. Zero means that the current part was created by insert rather than by merging other parts.

- `data_version` (UInt64) – Number that is used to determine which mutations should be applied to the data part (mutations with a version higher than `data_version`).
- `primary_key_bytes_in_memory` (UInt64) – The amount of memory (in bytes) used by primary key values.
- `primary_key_bytes_in_memory_allocated` (UInt64) – The amount of memory (in bytes) reserved for primary key values.
- `is_frozen` (UInt8) – Flag that shows that a partition data backup exists. 1, the backup exists. 0, the backup doesn't exist. For more details, see [FREEZE PARTITION](#)
- `database` (String) – Name of the database.
- `table` (String) – Name of the table.
- `engine` (String) – Name of the table engine without parameters.
- `path` (String) – Absolute path to the folder with data part files.
- `disk` (String) – Name of a disk that stores the data part.
- `hash_of_all_files` (String) – [sipHash128](#) 的压缩文件。
- `hash_of_uncompressed_files` (String) – [sipHash128](#) 未压缩的文件（带标记的文件，索引文件等。）。
- `uncompressed_hash_of_compressed_files` (String) – [sipHash128](#) 压缩文件中的数据，就好像它们是未压缩的。
- `bytes` (UInt64) – Alias for `bytes_on_disk`.
- `marks_size` (UInt64) – Alias for `marks_bytes`.

系统。`part_log`

该 `system.part_log` 表只有当创建 `part_log` 指定了服务器设置。

此表包含与以下情况发生的事件有关的信息 [数据部分](#) 在 [MergeTree](#) 家庭表，例如添加或合并数据。

该 `system.part_log` 表包含以下列：

- `event_type` (Enum) — Type of the event that occurred with the data part. Can have one of the following values:
 - `NEW_PART` — Inserting of a new data part.
 - `MERGE_PARTS` — Merging of data parts.
 - `DOWNLOAD_PART` — Downloading a data part.
 - `REMOVE_PART` — Removing or detaching a data part using [DETACH PARTITION](#).
 - `MUTATE_PART` — Mutating of a data part.
 - `MOVE_PART` — Moving the data part from the one disk to another one.
- `event_date` (Date) — Event date.
- `event_time` (DateTime) — Event time.
- `duration_ms` (UInt64) — Duration.
- `database` (String) — Name of the database the data part is in.
- `table` (String) — Name of the table the data part is in.
- `part_name` (String) — Name of the data part.
- `partition_id` (String) — ID of the partition that the data part was inserted to. The column takes the 'all'值，如果分区是由 `tuple()`。
- `rows` (UInt64) — The number of rows in the data part.
- `size_in_bytes` (UInt64) — Size of the data part in bytes.
- `merged_from` (Array(String)) — An array of names of the parts which the current part was made up from (after the merge).
- `bytes_uncompressed` (UInt64) — Size of uncompressed bytes.
- `read_rows` (UInt64) — The number of rows was read during the merge

- `read_rows` (UInt64) — The number of rows was read during the merge.
- `read_bytes` (UInt64) — The number of bytes was read during the merge.
- `error` (UInt16) — The code number of the occurred error.
- `exception` (String) — Text message of the occurred error.

该 `system.part_log` 表的第一个插入数据到后创建 `MergeTree` 桌子

系统。流程

该系统表用于实现 `SHOW PROCESSLIST` 查询。

列:

- `user` (String) – The user who made the query. Keep in mind that for distributed processing, queries are sent to remote servers under the `default` 用户。该字段包含特定查询的用户名，而不是此查询启动的查询的用户名。
- `address` (String) – The IP address the request was made from. The same for distributed processing. To track where a distributed query was originally made from, look at `system.processes` 查询请求者服务器上。
- `elapsed` (Float64) – The time in seconds since request execution started.
- `rows_read` (UInt64) – The number of rows read from the table. For distributed processing, on the requestor server, this is the total for all remote servers.
- `bytes_read` (UInt64) – The number of uncompressed bytes read from the table. For distributed processing, on the requestor server, this is the total for all remote servers.
- `total_rows_approx` (UInt64) – The approximation of the total number of rows that should be read. For distributed processing, on the requestor server, this is the total for all remote servers. It can be updated during request processing, when new sources to process become known.
- `memory_usage` (UInt64) – Amount of RAM the request uses. It might not include some types of dedicated memory. See the `max_memory_usage` 设置。
- `query` (String) – The query text. For `INSERT`，它不包括要插入的数据。
- `query_id` (String) – Query ID, if defined.

系统。text_log

包含日志记录条目。进入该表的日志记录级别可以通过以下方式进行限制 `text_log.level` 服务器设置。

列:

- `event_date` (Date)-条目的日期。
- `event_time` (DateTime)-条目的时间。
- `microseconds` (UInt32) -条目的微秒。
- `thread_name` (String) — Name of the thread from which the logging was done.
- `thread_id` (UInt64) — OS thread ID.
- `level` (Enum8) -入门级。
 - 'Fatal' = 1
 - 'Critical' = 2
 - 'Error' = 3
 - 'Warning' = 4
 - 'Notice' = 5
 - 'Information' = 6
 - 'Debug' = 7
 - 'Trace' = 8
- `query_id` (String)-查询的ID。
- `logger_name` (LowCardinality(String)) - Name of the logger (i.e. `DDLWorker`)
- `message` (String) -消息本身。
- `revision` (UInt32)-ClickHouse修订。
- `source_file` (LowCardinality(String))-从中完成日志记录的源文件。
- `source_line` (UInt64)-从中完成日志记录的源代码行。

系统。query_log

包含有关查询执行的信息。对于每个查询，您可以看到处理开始时间，处理持续时间，错误消息和其他信息。

注

该表不包含以下内容的输入数据 `INSERT` 查询。

ClickHouse仅在以下情况下创建此表 `query_log` 指定服务器参数。此参数设置日志记录规则，例如日志记录间隔或将记录查询的表的名称。

要启用查询日志记录，请设置 `log_queries` 参数为1。有关详细信息，请参阅 [设置](#) 科。

该 `system.query_log` 表注册两种查询：

1. 客户端直接运行的初始查询。
2. 由其他查询启动的子查询（用于分布式查询执行）。对于这些类型的查询，有关父查询的信息显示在 `initial_*` 列。

列：

- `type` (Enum8) — Type of event that occurred when executing the query. Values:
 - 'QueryStart' = 1 — Successful start of query execution.
 - 'QueryFinish' = 2 — Successful end of query execution.
 - 'ExceptionBeforeStart' = 3 — Exception before the start of query execution.
 - 'ExceptionWhileProcessing' = 4 — Exception during the query execution.
- `event_date` (Date) — Query starting date.
- `event_time` (DateTime) — Query starting time.
- `query_start_time` (DateTime) — Start time of query execution.
- `query_duration_ms` (UInt64) — Duration of query execution.
- `read_rows` (UInt64) — Number of read rows.
- `read_bytes` (UInt64) — Number of read bytes.
- `written_rows` (UInt64) — For `INSERT` 查询，写入的行数。对于其他查询，列值为0。
- `written_bytes` (UInt64) — For `INSERT` 查询时，写入的字节数。对于其他查询，列值为0。
- `result_rows` (UInt64) — Number of rows in the result.
- `result_bytes` (UInt64) — Number of bytes in the result.
- `memory_usage` (UInt64) — Memory consumption by the query.
- `query` (String) — Query string.
- `exception` (String) — Exception message.
- `stack_trace` (String) — Stack trace (a list of methods called before the error occurred). An empty string, if the query is completed successfully.
- `is_initial_query` (UInt8) — Query type. Possible values:
 - 1 — Query was initiated by the client.
 - 0 — Query was initiated by another query for distributed query execution.
- `user` (String) — Name of the user who initiated the current query.
- `query_id` (String) — ID of the query.
- `address` (IPv6) — IP address that was used to make the query.
- `port` (UInt16) — The client port that was used to make the query.
- `initial_user` (String) — Name of the user who ran the initial query (for distributed query execution).
- `initial_query_id` (String) — ID of the initial query (for distributed query execution).
- `initial_address` (IPv6) — IP address that the parent query was launched from.
- `initial_port` (UInt16) — The client port that was used to make the parent query.
- `interface` (UInt8) — Interface that the query was initiated from. Possible values:
 - 1 — TCP.
 - 2 — HTTP.
- `os_user` (String) — OS's username who runs `客户端`。
- `client_hostname` (String) — Hostname of the client machine where the `客户端` 或者运行另一个

TCP客户端。

- `client_name` (String) — The 嘉ツ徳 client 嘉ツ徳 或另一个TCP客户端名称。
- `client_revision` (UInt32) — Revision of the 嘉ツ徳 client 嘉ツ徳 或另一个TCP客户端。
- `client_version_major` (UInt32) — Major version of the 嘉ツ徳 client 嘉ツ徳 或另一个TCP客户端。
- `client_version_minor` (UInt32) — Minor version of the 嘉ツ徳 client 嘉ツ徳 或另一个TCP客户端。
- `client_version_patch` (UInt32) — Patch component of the 嘉ツ徳 client 嘉ツ徳 或另一个TCP客户端版本。
- `http_method` (UInt8) — HTTP method that initiated the query. Possible values:
 - 0 — The query was launched from the TCP interface.
 - 1 — GET 方法被使用。
 - 2 — POST 方法被使用。
- `http_user_agent` (String) — The UserAgent http请求中传递的标头。
- `quota_key` (String) — The “quota key” 在指定 配额 设置 (见 `keyed`).
- `revision` (UInt32) — ClickHouse revision.
- `thread_numbers` (Array(UInt32)) — Number of threads that are participating in query execution.
- `ProfileEvents.Names` (Array(String)) — Counters that measure different metrics. The description of them could be found in the table 系统。活动
- `ProfileEvents.Values` (Array(UInt64)) — Values of metrics that are listed in the `ProfileEvents.Names` 列。
- `Settings.Names` (Array(String)) — Names of settings that were changed when the client ran the query. To enable logging changes to settings, set the `log_query_settings` 参数为1。
- `Settings.Values` (Array(String)) — Values of settings that are listed in the `Settings.Names` 列。

每个查询创建一个或两个行中 `query_log` 表，具体取决于查询的状态：

1. 如果查询执行成功，将创建两个类型为1和2的事件（请参阅 `type` 列）。
2. 如果在查询处理过程中发生错误，将创建两个类型为1和4的事件。
3. 如果在启动查询之前发生错误，将创建类型为3的单个事件。

默认情况下，日志以7.5秒的间隔添加到表中。您可以在设置此时间间隔 `query_log` 服务器设置（请参阅 `flush_interval_milliseconds` 参数）。要强制将日志从内存缓冲区刷新到表中，请使用 `SYSTEM FLUSH LOGS` 查询。

当手动删除表时，它将自动动态创建。请注意，所有以前的日志将被删除。

注

日志的存储周期是无限的。日志不会自动从表中删除。您需要自己组织删除过时的日志。

您可以指定一个任意的分区键 `system.query_log` 表中的 `query_log` 服务器设置（请参阅 `partition_by` 参数）。

系统。`query_thread_log`

该表包含有关每个查询执行线程的信息。

ClickHouse仅在以下情况下创建此表 `query_thread_log` 指定服务器参数。此参数设置日志记录规则，例如日志记录间隔或记录查询的表的名称。

要启用查询日志记录，请设置 `log_query_threads` 参数为1。有关详细信息，请参阅 [设置](#) 科。

列:

- `event_date` (Date) — the date when the thread has finished execution of the query.
- `event_time` (DateTime) — the date and time when the thread has finished execution of the query.
- `query_start_time` (DateTime) — Start time of query execution.
- `query_duration_ms` (UInt64) — Duration of query execution.
- `read_rows` (UInt64) — Number of read rows.
- `read_bytes` (UInt64) — Number of read bytes.
- `written_rows` (UInt64) — For `INSERT` 查询，写入的行数。对于其他查询，列值为0。
- `written_bytes` (UInt64) — For `INSERT` 查询时，写入的字节数。对于其他查询，列值为0。

- `memory_usage` (Int64) — The difference between the amount of allocated and freed memory in context of this thread.
- `peak_memory_usage` (Int64) — The maximum difference between the amount of allocated and freed memory in context of this thread.
- `thread_name` (String) — Name of the thread.
- `thread_number` (UInt32) — Internal thread ID.
- `os_thread_id` (Int32) — OS thread ID.
- `master_thread_id` (UInt64) — OS initial ID of initial thread.
- `query` (String) — Query string.
- `is_initial_query` (UInt8) — Query type. Possible values:
 - 1 — Query was initiated by the client.
 - 0 — Query was initiated by another query for distributed query execution.
- `user` (String) — Name of the user who initiated the current query.
- `query_id` (String) — ID of the query.
- `address` (IPv6) — IP address that was used to make the query.
- `port` (UInt16) — The client port that was used to make the query.
- `initial_user` (String) — Name of the user who ran the initial query (for distributed query execution).
- `initial_query_id` (String) — ID of the initial query (for distributed query execution).
- `initial_address` (IPv6) — IP address that the parent query was launched from.
- `initial_port` (UInt16) — The client port that was used to make the parent query.
- `interface` (UInt8) — Interface that the query was initiated from. Possible values:
 - 1 — TCP.
 - 2 — HTTP.
- `os_user` (String) — OS's username who runs `client`.
- `client_hostname` (String) — Hostname of the client machine where the `client` 或者运行另一个 TCP 客户端。
- `client_name` (String) — The `client` 或者另一个 TCP 客户端名称。
- `client_revision` (UInt32) — Revision of the `client` 或者另一个 TCP 客户端。
- `client_version_major` (UInt32) — Major version of the `client` 或者另一个 TCP 客户端。
- `client_version_minor` (UInt32) — Minor version of the `client` 或者另一个 TCP 客户端。
- `client_version_patch` (UInt32) — Patch component of the `client` 或者另一个 TCP 客户端版本。
- `http_method` (UInt8) — HTTP method that initiated the query. Possible values:
 - 0 — The query was launched from the TCP interface.
 - 1 — GET 方法被使用。
 - 2 — POST 方法被使用。
- `http_user_agent` (String) — The UserAgent http 请求中传递的标头。
- `quota_key` (String) — The “quota key” 在指定 `keyed` 设置 (见 `keyed`).
- `revision` (UInt32) — ClickHouse revision.
- `ProfileEvents.Names` (Array(String)) — Counters that measure different metrics for this thread. The description of them could be found in the table 系统。活动
- `ProfileEvents.Values` (Array(UInt64)) — Values of metrics for this thread that are listed in the `ProfileEvents.Names` 列。

默认情况下，日志以 7.5 秒的间隔添加到表中。您可以在设置此时间间隔 `query_thread_log` 服务器设置（请参阅 `flush_interval_milliseconds` 参数）。要强制将日志从内存缓冲区刷新到表中，请使用 `SYSTEM FLUSH LOGS` 查询。

当手动删除表时，它将自动动态创建。请注意，所有以前的日志将被删除。

注

日志的存储周期是无限的。日志不会自动从表中删除。您需要自己组织删除过时的日志。

您可以指定一个任意的分区键 `system.query_thread_log` 表中的 `query_thread_log` 服务器设置（请参阅 `partition_by` 参数）。

系统。trace_log

包含采样查询探查器收集的堆栈跟踪。

ClickHouse创建此表时 `trace_log` 服务器配置部分被设置。也是 `query_profiler_real_time_period_ns` 和 `query_profiler_cpu_time_period_ns` 应设置设置。

要分析日志，请使用 `addressToLine`, `addressToSymbol` 和 `demangle` 内省功能。

列:

- `event_date` (`日期`) — Date of sampling moment.
- `event_time` (`日期时间`) — Timestamp of the sampling moment.
- `timestamp_ns` (`UInt64`) — Timestamp of the sampling moment in nanoseconds.
- `revision` (`UInt32`) — ClickHouse server build revision.

通过以下方式连接到服务器 `clickhouse-client`，你看到的字符串类似于 `Connected to ClickHouse server version 19.18.1 revision 54429..` 该字段包含 `revision`，但不是 `version` 的服务器。

- `timer_type` (`枚举8`) — Timer type:
 - `Real` 表示挂钟时间。
 - `CPU` 表示CPU时间。
- `thread_number` (`UInt32`) — Thread identifier.
- `query_id` (`字符串`) — Query identifier that can be used to get details about a query that was running from the `query_log` 系统表。
- `trace` (`数组(UInt64)`) — Stack trace at the moment of sampling. Each element is a virtual memory address inside ClickHouse server process.

示例

```
SELECT * FROM system.trace_log LIMIT 1 \G
```

Row 1:

```
event_date: 2019-11-15
event_time: 2019-11-15 15:09:38
revision: 54428
timer_type: Real
thread_number: 48
query_id: acc4d61f-5bd1-4a3e-bc91-2180be37c915
trace:
[94222141367858,94222152240175,94222152325351,94222152329944,94222152330796,94222151449980,9422214088167,94222151682763,94222144088167,94222151682763,94222144088167,94222144058283,94222144059248,94222091840750,94222091842302,94222091831228,94222189631488,140509950166747,140509942945935]
```

系统。副本

包含驻留在本地服务器上的复制表的信息和状态。

此表可用于监视。该表对于每个已复制的*表都包含一行。

示例:

```
SELECT *
FROM system.replicas
```

```
FROM system.replicas
WHERE table = 'visits'
FORMAT Vertical
```

Row 1:

```
database:          merge
table:            visits
engine:           ReplicatedCollapsingMergeTree
is_leader:        1
can_become_leader: 1
is_READONLY:      0
is_SESSION_EXPIRED: 0
future_parts:     1
parts_to_check:   0
zookeeper_path:  /clickhouse/tables/01-06/visits
replica_name:    example01-06-1.yandex.ru
replica_path:    /clickhouse/tables/01-06/visits/replicas/example01-06-1.yandex.ru
columns_version: 9
queue_size:       1
inserts_in_queue: 0
merges_in_queue:  1
part_mutations_in_queue: 0
queue_oldest_time: 2020-02-20 08:34:30
inserts_oldest_time: 0000-00-00 00:00:00
merges_oldest_time: 2020-02-20 08:34:30
part_mutations_oldest_time: 0000-00-00 00:00:00
oldest_part_to_get:
oldest_part_to_merge_to: 20200220_20284_20840_7
oldest_part_to_mutate_to:
log_max_index:      596273
log_pointer:        596274
last_queue_update:  2020-02-20 08:34:32
absolute_delay:     0
total_replicas:    2
active_replicas:   2
```

列:

- **database (String)**-数据库名称
- **table (String)**-表名
- **engine (String)**-表引擎名称
- **is_leader (UInt8)**-副本是否是领导者。
一次只有一个副本可以成为领导者。领导者负责选择要执行的后台合并。
- 请注意，可以对任何可用且在ZK中具有会话的副本执行写操作，而不管该副本是否为leader。
- **can_become_leader (UInt8)**-副本是否可以当选为领导者。
- **is_READONLY (UInt8)**-副本是否处于只读模式。
如果配置没有ZooKeeper的部分，如果在ZooKeeper中重新初始化会话时发生未知错误，以及在ZooKeeper中重新初始化会话时发生未知错误，则此模式将打开。
- **is_SESSION_EXPIRED (UInt8)**-与ZooKeeper的会话已经过期。基本上一样 **is_READONLY**.
- **future_parts (UInt32)**-由于尚未完成的插入或合并而显示的数据部分的数量。
- **parts_to_check (UInt32)**-队列中用于验证的数据部分的数量。如果怀疑零件可能已损坏，则将其放入验证队列。
- **zookeeper_path (String)**-在ZooKeeper中的表数据路径。
- **replica_name (String)**-在动物园管理员副本名称。同一表的不同副本具有不同的名称。
- **replica_path (String)**-在ZooKeeper中的副本数据的路径。与连接相同 'zookeeper_path/replicas/replica_path'.
- **columns_version (Int32)**-表结构的版本号。指示执行ALTER的次数。如果副本有不同的版本，这意味着一些副本还没有做出所有的改变。
- **queue_size (UInt32)**-等待执行的操作的队列大小。操作包括插入数据块、合并和某些其他操作。它通常与

- `future_parts`.
- `inserts_in_queue` (`UInt32`) - 需要插入数据块的数量。插入通常复制得相当快。如果这个数字很大，这意味着有什么不对劲。
 - `merges_in_queue` (`UInt32`) - 等待进行合并的数量。有时合并时间很长，因此此值可能长时间大于零。
 - `part_mutations_in_queue` (`UInt32`) - 等待进行的突变的数量。
 - `queue_oldest_time` (`DateTime`) - 如果 `queue_size` 大于 0，显示何时将最旧的操作添加到队列中。
 - `inserts_oldest_time` (`DateTime`) - 看 `queue_oldest_time`
 - `merges_oldest_time` (`DateTime`) - 看 `queue_oldest_time`
 - `part_mutations_oldest_time` (`DateTime`) - 看 `queue_oldest_time`

接下来的4列只有在有ZK活动会话的情况下才具有非零值。

- `log_max_index` (`UInt64`) - 一般活动日志中的最大条目数。
- `log_pointer` (`UInt64`) - 副本复制到其执行队列的常规活动日志中的最大条目数加一。如果 `log_pointer` 比 `log_max_index`，有点不对劲。
- `last_queue_update` (`DateTime`) - 上次更新队列时。
- `absolute_delay` (`UInt64`) - 当前副本有多大滞后秒。
- `total_replicas` (`UInt8`) - 此表的已知副本总数。
- `active_replicas` (`UInt8`) - 在ZooKeeper中具有会话的此表的副本的数量（即正常运行的副本的数量）。

如果您请求所有列，表可能会工作得有点慢，因为每行都会从ZooKeeper进行几次读取。

如果您没有请求最后4列 (`log_max_index`, `log_pointer`, `total_replicas`, `active_replicas`)，表工作得很快。

例如，您可以检查一切是否正常工作，如下所示：

```

SELECT
    database,
    table,
    is_leader,
    is_readonly,
    is_session_expired,
    future_parts,
    parts_to_check,
    columns_version,
    queue_size,
    inserts_in_queue,
    merges_in_queue,
    log_max_index,
    log_pointer,
    total_replicas,
    active_replicas
FROM system.replicas
WHERE
    is_READONLY
    OR is_SESSION_EXPIRED
    OR future_parts > 20
    OR parts_to_check > 10
    OR queue_size > 20
    OR inserts_in_queue > 10
    OR log_max_index - log_pointer > 10
    OR total_replicas < 2
    OR active_replicas < total_replicas
  
```

如果这个查询没有返回任何东西，这意味着一切都很好。

系统。设置

包含有关当前用户的会话设置的信息。

列：

- `name` (字符串) — Setting name.
- `value` (字符串) — Setting value.
- `changed` (UInt8) — Shows whether a setting is changed from its default value.
- `description` (字符串) — Short setting description.
- `min` (可为空(字符串)) — Minimum value of the setting, if any is set via [制约因素](#). 如果设置没有最小值，则包含 `NULL`.
- `max` (可为空(字符串)) — Maximum value of the setting, if any is set via [制约因素](#). 如果设置没有最大值，则包含 `NULL`.
- `readonly` (UInt8) — Shows whether the current user can change the setting:
 - 0 — Current user can change the setting.
 - 1 — Current user can't change the setting.

示例

下面的示例演示如何获取有关名称包含的设置的信息 `min_i`.

```
SELECT *
FROM system.settings
WHERE name LIKE '%min_i%'
```

name	value	changed	description
	min	max	readonly
min_insert_block_size_rows	1048576	0	Squash blocks passed to INSERT query to specified size in rows, if blocks are not big enough.
min_insert_block_size_bytes	268435456	0	Squash blocks passed to INSERT query to specified size in bytes, if blocks are not big enough.
read_backoff_min_interval_between_events_ms	1000	0	Settings to reduce the number of threads in case of slow reads. Do not pay attention to the event, if the previous one has passed less than a certain amount of time.
NULL	NULL	0	

使用 `WHERE changed` 可以是有用的，例如，当你想检查：

- 配置文件中的设置是否正确加载并正在使用。
- 在当前会话中更改的设置。

```
SELECT * FROM system.settings WHERE changed AND name='load_balancing'
```

另请参阅

- [设置](#)
- [查询权限](#)
- [对设置的限制](#)

系统。表_engines

name	value
max_threads	8
use_uncompressed_cache	0
load_balancing	random
max_memory_usage	100000000000

系统。merge_tree_settings

包含有关以下设置的信息 MergeTree 桌子

列:

- `name` (String) — Setting name.
- `value` (String) — Setting value.
- `description` (String) — Setting description.
- `type` (String) — Setting type (implementation specific string value).
- `changed` (UInt8) — Whether the setting was explicitly defined in the config or explicitly changed.

系统。表_engines

包含服务器支持的表引擎的描述及其功能支持信息。

此表包含以下列 (列类型显示在括号中):

- `name` (String) — The name of table engine.
- `supports_settings` (UInt8) — Flag that indicates if table engine supports SETTINGS 条款
- `supports_skipping_indices` (UInt8) — Flag that indicates if table engine supports 跳过索引.
- `supports_ttl` (UInt8) — Flag that indicates if table engine supports TTL.
- `supports_sort_order` (UInt8) — Flag that indicates if table engine supports clauses PARTITION_BY, PRIMARY_KEY, ORDER_BY 和 SAMPLE_BY.
- `supports_replication` (UInt8) — Flag that indicates if table engine supports 数据复制.
- `supports_deduplication` (UInt8) — Flag that indicates if table engine supports data deduplication.

示例:

```
SELECT *
FROM system.table_engines
WHERE name in ('Kafka', 'MergeTree', 'ReplicatedCollapsingMergeTree')
```

name	supports_settings	supports_skipping_indices	supports_sort_order	supports_ttl	supports_replication	supports_deduplication
Kafka	1	0	0	1	1	0
MergeTree	1	1	1	0	0	0
ReplicatedCollapsingMergeTree	1	1	1	0	0	1

另请参阅

- 梅树家族 [查询子句](#)
- 卡夫卡 [设置](#)
- 加入我们 [设置](#)

系统。表

包含服务器知道的每个表的元数据。分离的表不显示在 `system.tables`.

此表包含以下列 (列类型显示在括号中):

- `database` (String) — The name of the database the table is in.
- `name` (String) — Table name.

- `engine` (`String`) — Table engine name (without parameters).
- `is_temporary` (`UInt8`)-指示表是否是临时的标志。
- `data_path` (`String`)-文件系统中表数据的路径。
- `metadata_path` (`String`)-文件系统中表元数据的路径。
- `metadata_modification_time` (`DateTime`)-表元数据的最新修改时间。
- `dependencies_database` (数组(`字符串`))-数据库依赖关系.
- `dependencies_table` (数组 (`字符串`)) - 表依赖关系 (**MaterializedView** 基于当前表的表) 。
- `create_table_query` (`String`)-用于创建表的查询。
- `engine_full` (`String`)-表引擎的参数。
- `partition_key` (`String`)-表中指定的分区键表达式。
- `sorting_key` (`String`)-表中指定的排序键表达式。
- `primary_key` (`String`)-表中指定的主键表达式。
- `sampling_key` (`String`)-表中指定的采样键表达式。
- `storage_policy` (`字符串`)-存储策略:
 - **MergeTree**
 - 分布
- `total_rows` (`Nullable(UInt64)`)-总行数，如果可以快速确定表中的确切行数，否则 `Null` (包括内衣 `Buffer` 表) 。
- `total_bytes` (`Nullable(UInt64)`)-总字节数，如果可以快速确定存储表的确切字节数，否则 `Null` (不 包括任何底层存储) 。
 - If the table stores data on disk, returns used space on disk (i.e. compressed).
 - 如果表在内存中存储数据,返回在内存中使用的近似字节数.

该 `system.tables` 表中使用 `SHOW TABLES` 查询实现。

系统。动物园管理员

如果未配置ZooKeeper，则表不存在。 允许从配置中定义的ZooKeeper集群读取数据。

查询必须具有 ‘path’ WHERE子句中的平等条件。 这是ZooKeeper中您想要获取数据的孩子的路径。

查询 `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` 输出对所有孩子的数据 `/clickhouse` 节点。

要输出所有根节点的数据，`write path= '/'`.

如果在指定的路径 ‘path’ 不存在，将引发异常。

例:

- `name` (`String`) — The name of the node.
- `path` (`String`) — The path to the node.
- `value` (`String`) — Node value.
- `dataLength` (`Int32`) — Size of the value.
- `numChildren` (`Int32`) — Number of descendants.
- `czxid` (`Int64`) — ID of the transaction that created the node.
- `mzxid` (`Int64`) — ID of the transaction that last changed the node.
- `pzxid` (`Int64`) — ID of the transaction that last deleted or added descendants.
- `ctime` (`DateTime`) — Time of node creation.
- `mtime` (`DateTime`) — Time of the last modification of the node.

- `version` (Int32) — Node version: the number of times the node was changed.
- `cversion` (Int32) — Number of added or removed descendants.
- `aversion` (Int32) — Number of changes to the ACL.
- `ephemeralOwner` (Int64) — For ephemeral nodes, the ID of the session that owns this node.

示例:

```
SELECT *
FROM system.zookeeper
WHERE path = '/clickhouse/tables/01-08/visits/replicas'
FORMAT Vertical
```

Row 1:

```
name: example01-08-1.yandex.ru
value:
czxid: 932998691229
mxzid: 932998691229
ctime: 2015-03-27 16:49:51
mtime: 2015-03-27 16:49:51
version: 0
cversion: 47
aversion: 0
ephemeralOwner: 0
dataLength: 0
numChildren: 7
pzxid: 987021031383
path: /clickhouse/tables/01-08/visits/replicas
```

Row 2:

```
name: example01-08-2.yandex.ru
value:
czxid: 933002738135
mxzid: 933002738135
ctime: 2015-03-27 16:57:01
mtime: 2015-03-27 16:57:01
version: 0
cversion: 37
aversion: 0
ephemeralOwner: 0
dataLength: 0
numChildren: 7
pzxid: 987021252247
path: /clickhouse/tables/01-08/visits/replicas
```

系统。突变

该表包含以下信息 突变 MergeTree表及其进展。每个突变命令由一行表示。该表具有以下列:

数据库, 表 -应用突变的数据库和表的名称。

mutation_id -变异的ID 对于复制的表, 这些Id对应于znode中的名称 `<table_path_in_zookeeper>/mutations/` 动物园管理员的目录。对于未复制的表, Id对应于表的数据目录中的文件名。

命令 -Mutation命令字符串 (查询后的部分 `ALTER TABLE [db.]table`).

create_time -当这个突变命令被提交执行。

block_numbers partition_id block_numbers 组是 _端点列。对于复制表的空变, 它命令每个分区的一条记录。分

`BLOCK_NUMBERS.PARTITION_ID`, `BLOCK_NUMBERS.BLOCK_ID` - 分区ID和通过突变获取的块编号。在每个分区中，只有包含编号小于该分区中突变获取的块编号的块的在非复制表中，所有分区中的块编号形成一个序列。这意味着对于非复制表的突变，该列将包含一条记录，其中包含由突变获取的单个块编号。

parts_to_do - 为了完成突变，需要突变的数据部分的数量。

is_done - 变异完成了？？请注意，即使 `parts_to_do = 0` 由于长时间运行的INSERT将创建需要突变的新数据部分，因此可能尚未完成复制表的突变。

如果在改变某些部分时出现问题，以下列将包含其他信息：

latest_failed_part - 不能变异的最新部分的名称。

latest_fail_time - 最近的部分突变失败的时间。

latest_fail_reason - 导致最近部件变异失败的异常消息。

系统。磁盘

包含有关在定义的磁盘信息 [服务器配置](#)。

列：

- `name` ([字符串](#)) — Name of a disk in the server configuration.
- `path` ([字符串](#)) — Path to the mount point in the file system.
- `free_space` ([UInt64](#)) — Free space on disk in bytes.
- `total_space` ([UInt64](#)) — Disk volume in bytes.
- `keep_free_space` ([UInt64](#)) — Amount of disk space that should stay free on disk in bytes. Defined in the `keep_free_space_bytes` 磁盘配置参数。

系统。storage_policies

包含有关存储策略和卷中定义的信息 [服务器配置](#)。

列：

- `policy_name` ([字符串](#)) — Name of the storage policy.
- `volume_name` ([字符串](#)) — Volume name defined in the storage policy.
- `volume_priority` ([UInt64](#)) — Volume order number in the configuration.
- `disks` ([数组 \(字符串\)](#)) — Disk names, defined in the storage policy.
- `max_data_part_size` ([UInt64](#)) — Maximum size of a data part that can be stored on volume disks (0 — no limit).
- `move_factor` ([Float64](#)) — Ratio of free disk space. When the ratio exceeds the value of configuration parameter, ClickHouse start to move data to the next volume in order.

如果存储策略包含多个卷，则每个卷的信息将存储在表的单独行中。

如何使用ClickHouse测试您的硬件

使用此指令，您可以在任何服务器上运行基本的ClickHouse性能测试，而无需安装ClickHouse软件包。

1. 转到“commits”页数：<https://github.com/ClickHouse/ClickHouse/commits/master>
2. 点击第一个绿色复选标记或红色十字与绿色“ClickHouse Build Check”然后点击“Details”附近链接“ClickHouse Build Check”。在一些提交中没有这样的链接，例如与文档的提交。在这种情况下，请选择具有此链接的最近提交。
3. 将链接复制到“clickhouse”二进制为amd64或aarch64。
4. ssh到服务器并使用wget下载它：

```
# For amd64:  
wget https://clickhouse-builds.s3.yandex.net/0/00ba767f5d2a929394ea3be193b1f79074a1c4bc/1578163263_binary/clickhouse  
# For aarch64:  
wget https://clickhouse-builds.s3.yandex.net/0/00ba767f5d2a929394ea3be193b1f79074a1c4bc/1578161264_binary/clickhouse  
# Then do:  
chmod a+x clickhouse
```

1. 下载配置:

```
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/programs/server/config.xml  
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/programs/server/users.xml  
mkdir config.d  
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/programs/server/config.d/path.xml -O config.d/path.xml  
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/programs/server/config.d/log_to_console.xml -O config.d/log_to_console.xml
```

1. 下载基准测试文件:

```
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/benchmark/clickhouse/benchmark-new.sh  
chmod a+x benchmark-new.sh  
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/benchmark/clickhouse/queries.sql
```

1. 根据下载测试数据 [Yandex梅里卡数据集](#) 说明 (“hits” 表包含100万行)。

```
wget https://clickhouse-datasets.s3.yandex.net/hits/partitions/hits_100m_obfuscated_v1.tar.xz  
tar xvf hits_100m_obfuscated_v1.tar.xz -C .  
mv hits_100m_obfuscated_v1/* .
```

1. 运行服务器:

```
./clickhouse server
```

1. 检查数据 : ssh到另一个终端中的服务器

```
./clickhouse client --query "SELECT count() FROM hits_100m_obfuscated"  
100000000
```

1. 编辑benchmark-new.sh，改变 `clickhouse-client` 到 `./clickhouse client` 并添加 `--max_memory_usage 1000000000000` 参数。

```
mcedit benchmark-new.sh
```

1. 运行基准测试:

```
./benchmark-new.sh hits_100m_obfuscated
```

1. 将有关硬件配置的编号和信息发送到clickhouse-feedback@yandex-team.com

所有结果都在这里公布：<https://clickhouse.技术/基准/硬件/>

使用建议

CPU

必须支持SSE4.2指令集。现代处理器（自2008年以来）支持它。

选择处理器时，与较少的内核和较高的时钟速率相比，更喜欢大量内核和稍慢的时钟速率。
例如，具有2600MHz的16核心比具有3600MHz的8核心更好。

超线程

不要禁用超线程。它有助于某些查询，但不适用于其他查询。

涡轮增压

强烈推荐涡轮增压。它显着提高了典型负载的性能。

您可以使用 `turbostat` 要查看负载下的CPU的实际时钟速率。

CPU缩放调控器

始终使用 `performance` 缩放调控器。该 `on-demand` 随着需求的不断增加，缩放调节器的工作要糟糕得多。

```
echo 'performance' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

CPU限制

处理器可能会过热。使用 `dmesg` 看看CPU的时钟速率是否由于过热而受到限制。

此限制也可以在数据中心级别的外部设置。您可以使用 `turbostat` 在负载下监视它。

RAM

对于少量数据（高达-200GB压缩），最好使用与数据量一样多的内存。

对于大量数据和处理交互式（在线）查询时，应使用合理数量的RAM（128GB或更多），以便热数据子集适合页面缓存。
即使对于每台服务器约50TB的数据量，使用128GB的RAM与64GB相比显着提高了查询性能。

交换文件

始终禁用交换文件。不这样做的唯一原因是，如果您使用的ClickHouse在您的个人笔记本电脑。

巨大的页面

始终禁用透明巨大的页面。它会干扰内存分配，从而导致显着的性能下降。

```
echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

使用 `perf top` 观看内核中用于内存管理的时间。

永久巨大的页面也不需要被分配。

存储子系统

如果您的预算允许您使用SSD，请使用SSD。

如果没有，请使用硬盘。SATA硬盘7200转就行了。

优先选择带有本地硬盘驱动器的大量服务器，而不是带有附加磁盘架的小量服务器。
但是对于存储具有罕见查询的档案，货架将起作用。

RAID

当使用硬盘，你可以结合他们的RAID-10，RAID-5，RAID-6或RAID-50。

对于Linux，软件RAID更好（与 mdadm）。我们不建议使用LVM。

当创建RAID-10，选择 far 布局。

如果您的预算允许，请选择RAID-10。

如果您有超过4个磁盘，请使用RAID-6（首选）或RAID-50，而不是RAID-5。

当使用RAID-5、RAID-6或RAID-50时，始终增加stripe_cache_size，因为默认值通常不是最佳选择。

```
echo 4096 | sudo tee /sys/block/md2/md/stripe_cache_size
```

使用以下公式，从设备数量和块大小计算确切数量: $2 * \text{num_devices} * \text{chunk_size_in_bytes} / 4096$.

1025KB的块大小足以满足所有RAID配置。

切勿将块大小设置得太小或太大。

您可以在SSD上使用RAID-0。

无论使用何种RAID，始终使用复制来保证数据安全。

使用长队列启用NCQ。对于HDD，选择CFQ调度程序，对于SSD，选择noop。不要减少‘readahead’设置。

对于HDD，启用写入缓存。

文件系统

Ext4是最可靠的选择。设置挂载选项 noatime, nobarrier.

XFS也是合适的，但它还没有经过ClickHouse的彻底测试。

大多数其他文件系统也应该正常工作。具有延迟分配的文件系统工作得更好。

Linux内核

不要使用过时的Linux内核。

网络

如果您使用的是IPv6，请增加路由缓存的大小。

3.2之前的Linux内核在IPv6实现方面遇到了许多问题。

如果可能的话，至少使用一个10GB的网络。1Gb也可以工作，但对于使用数十tb的数据修补副本或处理具有大量中间数据的分布式查询，情况会更糟。

动物园管理员

您可能已经将ZooKeeper用于其他目的。您可以使用相同的zookeeper安装，如果它还没有超载。

It's best to use a fresh version of ZooKeeper – 3.4.9 or later. The version in stable Linux distributions may be outdated.

You should never use manually written scripts to transfer data between different ZooKeeper clusters, because the result will be incorrect for sequential nodes. Never use the «zkcopy» utility for the same reason: <https://github.com/ksprojects/zkcopy/issues/15>

如果要将现有ZooKeeper集群分为两个，正确的方法是增加其副本的数量，然后将其重新配置为两个独立的集群。

不要在与ClickHouse相同的服务器上运行ZooKeeper。由于ZooKeeper对延迟非常敏感，ClickHouse可能会利用所有可用的系统资源。

使用默认设置，ZooKeeper是一个定时炸弹：

使用默认配置时，ZooKeeper服务器不会从旧快照和日志中删除文件（请参阅autpurge），这是操作员的责任。

必须拆除炸弹

下面的ZooKeeper (3.5.1) 配置在Yandex中使用。梅地卡生产环境截至2017年5月20日：

动物园cfg:

```
## http://hadoop.apache.org/zookeeper/docs/current/zookeeperAdmin.html

## The number of milliseconds of each tick
tickTime=2000
## The number of ticks that the initial
## synchronization phase can take
initLimit=30000
## The number of ticks that can pass between
## sending a request and getting an acknowledgement
syncLimit=10

maxClientCnxns=2000

maxSessionTimeout=60000000
## the directory where the snapshot is stored.
dataDir=/opt/zookeeper/{{ cluster['name'] }}/data
## Place the dataLogDir to a separate physical disc for better performance
dataLogDir=/opt/zookeeper/{{ cluster['name'] }}/logs

autopurge.snapRetainCount=10
autopurge.purgeInterval=1

## To avoid seeks ZooKeeper allocates space in the transaction log file in
## blocks of preAllocSize kilobytes. The default block size is 64M. One reason
## for changing the size of the blocks is to reduce the block size if snapshots
## are taken more often. (Also, see snapCount).
preAllocSize=131072

## Clients can submit requests faster than ZooKeeper can process them,
## especially if there are a lot of clients. To prevent ZooKeeper from running
## out of memory due to queued requests, ZooKeeper will throttle clients so that
## there is no more than globalOutstandingLimit outstanding requests in the
## system. The default limit is 1,000. ZooKeeper logs transactions to a
## transaction log. After snapCount transactions are written to a log file a
## snapshot is started and a new transaction log file is started. The default
## snapCount is 10,000.
snapCount=3000000

## If this option is defined, requests will be logged to a trace file named
## traceFile.year.month.day.
##traceFile=

## Leader accepts client connections. Default value is "yes". The leader machine
## coordinates updates. For higher update throughput at the slight expense of
## read throughput the leader can be configured to not accept clients and focus
## on coordination.
leaderServes=yes

standaloneEnabled=false
dynamicConfigFile=/etc/zookeeper-{{ cluster['name'] }}/conf/zoo.cfg.dynamic
```

Java版本:

```
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

JVM参数:

```
NAME=zookeeper-{{ cluster['name'] }}
ZOOCFGDIR=/etc/$NAME/conf

## TODO this is really ugly
## How to find out, which jars are needed?
## seems, that log4j requires the log4j.properties file to be in the classpath
CLASSPATH="$ZOOCFGDIR:/usr/build/classes:/usr/build/lib/*.jar:/usr/share/zookeeper/zookeeper-3.5.1-
metrika.jar:/usr/share/zookeeper/slf4j-log4j12-1.7.5.jar:/usr/share/zookeeper/slf4j-api-
1.7.5.jar:/usr/share/zookeeper/servlet-api-2.5-20081211.jar:/usr/share/zookeeper/netty-
3.7.0.Final.jar:/usr/share/zookeeper/log4j-1.2.16.jar:/usr/share/zookeeper/jline-2.11.jar:/usr/share/zookeeper/jetty-util-
6.1.26.jar:/usr/share/zookeeper/jetty-6.1.26.jar:/usr/share/zookeeper/javacc.jar:/usr/share/zookeeper/jackson-mapper-
asl-1.9.11.jar:/usr/share/zookeeper/jackson-core-asl-1.9.11.jar:/usr/share/zookeeper/commons-cli-
1.2.jar:/usr/src/java/lib/*.jar:/usr/etc/zookeeper"

ZOOCFG="$ZOOCFGDIR/zoo.cfg"
ZOO_LOG_DIR=/var/log/$NAME
USER=zookeeper
GROUP=zookeeper
PDIR=/var/run/$NAME
PIDFILE=$PDIR/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
JAVA=/usr/bin/java
ZOMAIN="org.apache.zookeeper.server.quorum.QuorumPeerMain"
ZOO_LOG4J_PROP="INFO,ROLLINGFILE"
JMXLOCALONLY=false
JAVA_OPTS="-Xms{{ cluster.get('xms','128M') }} \
-Xmx{{ cluster.get('xmx','1G') }} \
-Xloggc:/var/log/$NAME/zookeeper-gc.log \
-XX:+UseGCLogFileRotation \
-XX:NumberOfGCLogFiles=16 \
-XX:GCLogFileSize=16M \
-verbose:gc \
-XX:+PrintGCTimeStamps \
-XX:+PrintGCDateStamps \
-XX:+PrintGCDetails \
-XX:+PrintTenuringDistribution \
-XX:+PrintGCApplicationStoppedTime \
-XX:+PrintGCApplicationConcurrentTime \
-XX:+PrintSafepointStatistics \
-XX:+UseParNewGC \
-XX:+UseConcMarkSweepGC \
-XX:+CMSParallelRemarkEnabled"
```

盐初始化:

```
description "zookeeper-{{ cluster['name'] }} centralized coordination service"

start on runlevel [2345]
stop on runlevel [!2345]

respawn

limit nofile 8192 8192

pre-start script
[ -r "/etc/zookeeper-{{ cluster['name'] }}/conf/environment" ] || exit 0
./etc/zookeeper-{{ cluster['name'] }}/conf/environment
```

```

[ -d $ZOO_LOG_DIR ] || mkdir -p $ZOO_LOG_DIR
chown $USER:$GROUP $ZOO_LOG_DIR
end script

script
./etc/zookeeper-{ { cluster['name'] } }/conf/environment
[ -r /etc/default/zookeeper ] && . /etc/default/zookeeper
if [ -z "$JMXDISABLE" ]; then
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.local.only=$JMXLOCALONLY"
fi
exec start-stop-daemon --start -c $USER --exec $JAVA --name zookeeper-{ { cluster['name'] } } \
-- -cp $CLASSPATH $JAVA_OPTS -Dzookeeper.log.dir=${ZOO_LOG_DIR} \
-Dzookeeper.root.logger=${ZOO_LOG4J_PROP} $ZOOMAIN $ZOOCFG
end script

```

操作

监控

可以监控到：

- 硬件资源的利用率。
- ClickHouse 服务的指标。

硬件资源利用率

ClickHouse 本身不会去监控硬件资源的状态。

强烈推荐监控以下监控项：

- 处理器上的负载和温度。

可以使用 [dmesg](<https://en.wikipedia.org/wiki/Dmesg>)，
[turbostat](<https://www.linux.org/docs/man8/turbostat.html>) 或者其他工具。

- 磁盘存储，RAM和网络的使用率。

ClickHouse 服务的指标。

ClickHouse服务本身具有用于自我状态监视指标。

要跟踪服务器事件，请观察服务器日志。请参阅配置文件的[logger] (`server_settings/settings.md#server_settings-logger`) 部分。

ClickHouse 收集的指标项：

- 服务于计算的资源占用的各种指标。
- 关于查询处理的常见统计信息。

可以在 系统。指标，系统。活动 以及 系统。asynchronous_metrics 等系统表查看所有的指标项。

可以配置ClickHouse 往 石墨 导入指标。参考 石墨部分 配置文件。在配置指标导出之前，需要参考Graphite 官方教程 搭建服务。

此外，您可以通过HTTP API 监视服务器可用性。将HTTP GET请求发送到 `/ping`。如果服务器可用，它将以 `200 OK` 响应。

要监视服务器集群的配置中，应设置`max_replica_delay_for_distributed_queries`参数并使用HTTP资源`/replicas_status`。如果副本可用，并且不延迟在其他副本之后，则对`/replicas_status`的请求将返回`200 OK`。如果副本滞后，请求将返回 `503 HTTP SERVICE UNAVAILABLE`，包括有关待办事项大小的信息。

访问权限

用户和访问权限在用户配置中设置。这通常是 `users.xml`。

用户被记录在 `users` 科。这里是一个片段 `users.xml` 文件：

```
<!-- Users and ACL. -->
<users>
    <!-- If the user name is not specified, the 'default' user is used. -->
    <default>
        <!-- Password could be specified in plaintext or in SHA256 (in hex format).

        If you want to specify password in plaintext (not recommended), place it in 'password' element.
        Example: <password>qwert</password>.
        Password could be empty.

        If you want to specify SHA256, place it in 'password_sha256_hex' element.
        Example:
<password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sha
256_hex>

        How to generate decent password:
        Execute: PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'
        In first line will be password and in second - corresponding SHA256.
-->
<password></password>

    <!-- A list of networks that access is allowed from.
        Each list item has one of the following forms:
        <ip> The IP address or subnet mask. For example: 198.51.100.0/24 or 2001:DB8::/32.
        <host> Host name. For example: example01. A DNS query is made for verification, and all addresses obtained
        are compared with the address of the customer.
        <host_regexp> Regular expression for host names. For example, ^example\d\d\d\d\d.yandex.ru$.
        To check it, a DNS PTR request is made for the client's address and a regular expression is applied to the
        result.
        Then another DNS query is made for the result of the PTR query, and all received address are compared to the
        client address.
        We strongly recommend that the regex ends with \.yandex\.ru$.

        If you are installing ClickHouse yourself, specify here:
        <networks>
            <ip>::/0</ip>
        </networks>
-->
<networks incl="networks" />

        <!-- Settings profile for the user. -->
        <profile>default</profile>

        <!-- Quota for the user. -->
        <quota>default</quota>
    </default>

    <!-- For requests from the Yandex.Metrica user interface via the API for data on specific counters. -->
    <web>
        <password></password>
        <networks incl="networks" />
        <profile>web</profile>
        <quota>default</quota>
        <allow_databases>
```

```
<allow_databases>
  <database>test</database>
</allow_databases>
</web>
```

您可以看到两个用户的声明: `default` 和 `web`. 我们添加了 `web` 用户分开。

该 `default` 在用户名未通过的情况下选择用户。该 `default` 如果服务器或群集的配置没有指定分布式查询处理，则 `user` 也用于分布式查询处理 `user` 和 `password`（见上的部分 **分布** 发动机）。

The user that is used for exchanging information between servers combined in a cluster must not have substantial restrictions or quotas – otherwise, distributed queries will fail.

密码以明文（不推荐）或 **SHA-256** 形式指定。哈希没有限制。在这方面，您不应将这些密码视为提供了针对潜在恶意攻击的安全性。相反，他们是必要的保护员工。

指定允许访问的网络列表。在此示例中，将从单独的文件加载两个用户的网络列表 (`/etc/metrika.xml`) 包含 `networks` 替代。这里是它的一个片段：

```
<yandex>
...
<networks>
  <ip>::/64</ip>
  <ip>203.0.113.0/24</ip>
  <ip>2001:DB8::/32</ip>
...
</networks>
</yandex>
```

您可以直接在以下内容中定义此网络列表 `users.xml`，或在文件中 `users.d` directory (for more information, see the section «[配置文件](#)»).

该配置包括解释如何从任何地方打开访问的注释。

对于在生产中使用，仅指定 `ip` 元素 (IP 地址及其掩码)，因为使用 `host` 和 `host_regex` 可能会导致额外的延迟。

Next the user settings profile is specified (see the section «[设置配置文件](#)»). You can specify the default profile, `default`. 配置文件可以有任何名称。您可以为不同的用户指定相同的配置文件。您可以在设置配置文件中编写的最重要的事情是 `readonly=1`，这确保只读访问。

Then specify the quota to be used (see the section «[配额](#)»). You can specify the default quota: `default`. It is set in the config by default to only count resource usage, without restricting it. The quota can have any name. You can specify the same quota for different users – in this case, resource usage is calculated for each user individually.

在可选 `<allow_databases>` 您还可以指定用户可以访问的数据库列表。默认情况下，所有数据库都可供用户使用。您可以指定 `default` 数据库。在这种情况下，默认情况下，用户将接收对数据库的访问权限。

访问 `system` 始终允许数据库（因为此数据库用于处理查询）。

用户可以通过以下方式获取其中所有数据库和表的列表 `SHOW` 查询或系统表，即使不允许访问单个数据库。

数据库访问是不相关的 **只读** 设置。您不能授予对一个数据库的完全访问权限，并 `readonly` 进入另一个。

配置文件

主服务器配置文件是 `config.xml`. 它驻留在 `/etc/clickhouse-server/` 目录。

单个设置可以在复盖 `*.xml` 和 `*.conf` 在文件 `conf.d` 和 `config.d` 配置文件旁边的目录。

该 `conf.d` 或 `config.d` 可以为许多配置文件的元素指定属性。

该 `replace` 或 `remove` 属性从这些属性的几条相似性。

如果两者都未指定，则递归组合元素的内容，替换重复子项的值。

如果 `replace` 如果指定，则将整个元素替换为指定的元素。

如果 `remove` 如果指定，则删除该元素。

The config can also define «substitutions». If an element has the `incl` 属性时，从文件中的相应替换将被用作该值。默认情况下，具有替换的文件的路径为 `/etc/metrika.xml`. 这可以在改变 包括 从服务器配置中的元素。替换值在指定 `/yandex/substitution_name` 这个文件中的元素。如果在指定的替换 `incl` 不存在，则将其记录在日志中。要防止ClickHouse记录丢失的替换，请指定 `optional="true"` 属性（例如，设置 宏 `server_settings/settings.md`）。

替换也可以从ZooKeeper执行。为此，请指定属性 `from_zk = "/path/to/node"`. 元素值被替换为节点的内容 `/path/to/node` 在动物园管理员。您还可以将整个XML子树放在ZooKeeper节点上，并将其完全插入到源元素中。

该 `config.xml` 文件可以指定具有用户设置、配置文件和配额的单独配置。这个配置的相对路径在 ‘`users_config`’ 元素。默认情况下，它是 `users.xml`. 如果 `users_config` 被省略，用户设置，配置文件和配额直接在指定 `config.xml`.

此外，`users_config` 可以从文件中复盖 `users_config.d` 目录（例如，`users.d`）和替换。例如，您可以为每个用户提供单独的配置文件，如下所示：

```
$ cat /etc/clickhouse-server/users.d/alice.xml
<yandex>
  <users>
    <alice>
      <profile>analytics</profile>
      <networks>
        <ip>::/0</ip>
      </networks>
      <password_sha256_hex>...</password_sha256_hex>
      <quota>analytics</quota>
    </alice>
  </users>
</yandex>
```

对于每个配置文件，服务器还会生成 `file-preprocessed.xml` 启动时的文件。这些文件包含所有已完成的替换和复盖，并且它们旨在提供信息。如果zookeeper替换在配置文件中使用，但ZooKeeper在服务器启动时不可用，则服务器将从预处理的文件中加载配置。

服务器跟踪配置文件中的更改，以及执行替换和复盖时使用的文件和ZooKeeper节点，并动态重新加载用户和集群的设置。这意味着您可以在不重新启动服务器的情况下修改群集、用户及其设置。

配额

配额允许您在一段时间内限制资源使用情况，或者只是跟踪资源的使用。

配额在用户配置中设置。这通常是 ‘`users.xml`’.

The system also has a feature for limiting the complexity of a single query. See the section «Restrictions on query complexity»).

与查询复杂性限制相比，配额：

- 对可以在一段时间内运行的一组查询设置限制，而不是限制单个查询。
- 占用在所有远程服务器上用于分布式查询处理的资源。

让我们来看看的部分 ‘`users.xml`’ 定义配额的文件。

```
<!-- Quotas -->
<quotas>
```

```

<!-- Quota name. -->
<default>
    <!-- Restrictions for a time period. You can set many intervals with different restrictions. -->
    <interval>
        <!-- Length of the interval. -->
        <duration>3600</duration>

        <!-- Unlimited. Just collect data for the specified time interval. -->
        <queries>0</queries>
        <errors>0</errors>
        <result_rows>0</result_rows>
        <read_rows>0</read_rows>
        <execution_time>0</execution_time>
    </interval>
</default>

```

默认情况下，配额只跟踪每小时的资源消耗，而不限制使用情况。

每次请求后，计算出的每个时间间隔的资源消耗将输出到服务器日志中。

```

<statbox>
    <!-- Restrictions for a time period. You can set many intervals with different restrictions. -->
    <interval>
        <!-- Length of the interval. -->
        <duration>3600</duration>

        <queries>1000</queries>
        <errors>100</errors>
        <result_rows>1000000000</result_rows>
        <read_rows>100000000000</read_rows>
        <execution_time>900</execution_time>
    </interval>

    <interval>
        <duration>86400</duration>

        <queries>10000</queries>
        <errors>1000</errors>
        <result_rows>5000000000</result_rows>
        <read_rows>500000000000</read_rows>
        <execution_time>7200</execution_time>
    </interval>
</statbox>

```

为‘statbox’配额，限制设置为每小时和每24小时（86,400秒）。时间间隔从实现定义的固定时刻开始计数。换句话说，24小时间隔不一定从午夜开始。

间隔结束时，将清除所有收集的值。在下一个小时内，配额计算将重新开始。

以下是可以限制的金额：

queries – The total number of requests.

errors – The number of queries that threw an exception.

result_rows – The total number of rows given as the result.

read_rows – The total number of source rows read from tables for running the query, on all remote servers.

execution_time – The total query execution time, in seconds (wall time).

如果在至少一个时间间隔内超出限制，则会引发异常，其中包含有关超出了哪个限制、哪个时间间隔以及新时间间隔开始时

(何时可以再次发送查询) 的文本。

Quotas can use the «quota key» feature in order to report on resources for multiple keys independently. Here is an example of this:

```
<!-- For the global reports designer. -->
<web_global>
  <!-- keyed - The quota_key "key" is passed in the query parameter,
       and the quota is tracked separately for each key value.
   For example, you can pass a Yandex.Metrica username as the key,
       so the quota will be counted separately for each username.
   Using keys makes sense only if quota_key is transmitted by the program, not by a user.

   You can also write <keyed_by_ip /> so the IP address is used as the quota key.
   (But keep in mind that users can change the IPv6 address fairly easily.)
-->
<keyed />
```

配额分配给用户 ‘users’ 部分的配置。参见 «Access rights» 部分。

对于分布式查询处理，累积的金额存储在请求者服务器上。因此如果用户去到另一个服务器，配额将从头开始。

服务器重新启动时，将重置配额。

服务器配置

`builtin_dictionaries_reload_interval`

重新加载内置字典的间隔时间（以秒为单位）。

ClickHouse每x秒重新加载内置字典。这使得编辑字典 “on the fly”，而无需重新启动服务器。

默认值:3600.

示例

```
<builtin_dictionaries_reload_interval>3600</builtin_dictionaries_reload_interval>
```

压缩

数据压缩配置 MergeTree 引擎表。

警告

如果您刚开始使用ClickHouse，请不要使用它。

配置模板:

```
<compression>
  <case>
    <min_part_size>...</min_part_size>
    <min_part_size_ratio>...</min_part_size_ratio>
    <method>...</method>
  </case>
```

```
...  
</compression>
```

<case> 参数:

- min_part_size – The minimum size of a data part.
- min_part_size_ratio – The ratio of the data part size to the table size.
- method – Compression method. Acceptable values: lz4 或 zstd.

您可以配置多个 <case> 部分。

满足条件时的操作:

- 如果数据部分与条件集匹配, ClickHouse 将使用指定的压缩方法。
- 如果数据部分匹配多个条件集, ClickHouse 将使用第一个匹配的条件集。

如果没有满足数据部分的条件, ClickHouse 使用 lz4 压缩。

示例

```
<compression incl="clickhouse_compression">  
  <case>  
    <min_part_size>10000000000</min_part_size>  
    <min_part_size_ratio>0.01</min_part_size_ratio>  
    <method>zstd</method>  
  </case>  
</compression>
```

default_database

默认数据库。

要获取数据库列表, 请使用 **SHOW DATABASES** 查询。

示例

```
<default_database>default</default_database>
```

default_profile

默认配置文件。

配置文件位于 user_config 参数指定的文件中。

示例

```
<default_profile>default</default_profile>
```

dictionaries_config

外部字典的配置文件的路径。

路径:

- 指定相对于服务器配置文件的绝对路径或路径。
- 路径可以包含通配符*和?.

另请参阅 “[外部字典](#)”。

示例

```
<dictionaries_config>*_dictionary.xml</dictionaries_config>
```

dictionaries_lazy_load

延迟加载字典。

如果 `true`，然后在第一次使用时创建每个字典。如果字典创建失败，则使用该字典的函数将引发异常。

如果 `false`，服务器启动时创建所有字典，如果出现错误，服务器将关闭。

默认值为 `true`.

示例

```
<dictionaries_lazy_load>true</dictionaries_lazy_load>
```

format_schema_path

包含输入数据方案的目录路径，例如输入数据的方案 [CapnProto](#) 格式。

示例

```
<!-- Directory containing schema files for various input formats. -->
<format_schema_path>format_schemas/</format_schema_path>
```

石墨

将数据发送到 [石墨](#)。

设置：

- host – The Graphite server.
- port – The port on the Graphite server.
- interval – The interval for sending, in seconds.
- timeout – The timeout for sending data, in seconds.
- root_path – Prefix for keys.
- metrics – Sending data from the [系统](#)。[指标](#) 桌子
- events – Sending deltas data accumulated for the time period from the [系统](#)。[活动](#) 桌子
- events_cumulative – Sending cumulative data from the [系统](#)。[活动](#) 桌子
- asynchronous_metrics – Sending data from the [系统](#)。[asynchronous_metrics](#) 桌子

您可以配置多个 `<graphite>` 条款 例如，您可以使用它以不同的时间间隔发送不同的数据。

示例

```
<graphite>
  <host>localhost</host>
  <port>42000</port>
  <timeout>0.1</timeout>
  <interval>60</interval>
  <root_path>one_min</root_path>
  <metrics>true</metrics>
  <events>true</events>
  <events_cumulative>false</events_cumulative>
  <asynchronous_metrics>true</asynchronous_metrics>
</graphite>
```

graphite_rollup

石墨细化数据的设置。

有关详细信息，请参阅 [GraphiteMergeTree](#).

示例

```
<graphite_rollup_example>
<default>
    <function>max</function>
    <retention>
        <age>0</age>
        <precision>60</precision>
    </retention>
    <retention>
        <age>3600</age>
        <precision>300</precision>
    </retention>
    <retention>
        <age>86400</age>
        <precision>3600</precision>
    </retention>
</default>
</graphite_rollup_example>
```

http_port/https_port

通过HTTP连接到服务器的端口。

如果 `https_port` 被指定, [openSSL](#) 必须配置。

如果 `http_port` 指定时, 即使设置了OpenSSL配置, 也会忽略该配置。

示例

```
<https_port>9999</https_port>
```

http_server_default_response

访问ClickHouse HTTP(s)服务器时默认显示的页面。

默认值为 “Ok.” (最后有换行符)

示例

打开 `https://tabix.io/` 访问时 `http://localhost: http_port.`

```
<http_server_default_response>
<![CDATA[<html ng-app="SMI2"><head><base href="http://ui.tabix.io/"></head><body><div ui-view="">
</div><script src="http://loader.tabix.io/master.js"></script></body></html>]]>
</http_server_default_response>
```

包括_从

带替换的文件的路径。

有关详细信息, 请参阅部分 “[配置文件](#)”.

示例

```
<include_from>/etc/metrica.xml</include_from>
```

interserver_http_port

用于在ClickHouse服务器之间交换数据的端口。

示例

```
<interserver_http_port>9009</interserver_http_port>
```

interserver_http_host

其他服务器可用于访问此服务器的主机名。

如果省略，它以相同的方式作为定义 `hostname-f` 指挥部

用于脱离特定的网络接口。

示例

```
<interserver_http_host>example.yandex.ru</interserver_http_host>
```

interserver_http_credentials

用户名和密码用于在以下期间进行身份验证 [复制](#) 与[复制*](#)引擎。这些凭据仅用于副本之间的通信，与ClickHouse客户端的凭据无关。服务器正在检查这些凭据以连接副本，并在连接到其他副本时使用相同的凭据。因此，这些凭据应该为集群中的所有副本设置相同。

默认情况下，不使用身份验证。

本节包含以下参数：

- `user` — `username`.
- `password` — `password`.

示例

```
<interserver_http_credentials>
  <user>admin</user>
  <password>222</password>
</interserver_http_credentials>
```

keep_alive_timeout

ClickHouse在关闭连接之前等待传入请求的秒数。默认为3秒。

示例

```
<keep_alive_timeout>3</keep_alive_timeout>
```

listen_host

对请求可能来自的主机的限制。如果您希望服务器回答所有这些问题，请指定 `::`。

例：

```
<listen_host>::1</listen_host>
<listen_host>127.0.0.1</listen_host>
```

记录器

日志记录设置。

键:

- level – Logging level. Acceptable values: trace, debug, information, warning, error.
- log – The log file. Contains all the entries according to level.
- errorlog – Error log file.
- size – Size of the file. Applies to log 和 errorlog. 一旦文件到达 size，ClickHouse存档并重命名它，并在其位置创建一个新的日志文件。
- count – The number of archived log files that ClickHouse stores.

示例

```
<logger>
  <level>trace</level>
  <log>/var/log/clickhouse-server/clickhouse-server.log</log>
  <errorlog>/var/log/clickhouse-server/clickhouse-server.err.log</errorlog>
  <size>1000M</size>
  <count>10</count>
</logger>
```

还支持写入系统日志。配置示例:

```
<logger>
  <use_syslog>1</use_syslog>
  <syslog>
    <address>syslog.remote:10514</address>
    <hostname>myhost.local</hostname>
    <facility>LOG_LOCAL6</facility>
    <format>syslog</format>
  </syslog>
</logger>
```

键:

- use_syslog — Required setting if you want to write to the syslog.
- address — The host[:port] of syslogd. If omitted, the local daemon is used.
- hostname — Optional. The name of the host that logs are sent from.
- facility — 系统日志工具关键字 在大写字母与 “LOG_” 前缀: (LOG_USER, LOG_DAEMON, LOG_LOCAL3, 等等)。默认值: LOG_USER 如果 address 被指定, LOG_DAEMON otherwise.
- format - Message format. Possible values: bsd 和 syslog.

宏

复制表的参数替换。

如果不使用复制的表，则可以省略。

有关详细信息，请参阅部分 “[创建复制的表](#)”。

示例

```
<macros incl="macros" optional="true" />
```

mark_cache_size

表引擎使用的标记缓存的近似大小（以字节为单位）**MergeTree** 家人

缓存为服务器共享，并根据需要分配内存。缓存大小必须至少为5368709120。

示例

```
<mark_cache_size>5368709120</mark_cache_size>
```

max_concurrent_queries

同时处理的请求的最大数量。

示例

```
<max_concurrent_queries>100</max_concurrent_queries>
```

max_connections

入站连接的最大数量。

示例

```
<max_connections>4096</max_connections>
```

max_open_files

打开文件的最大数量。

默认情况下: `maximum`.

我们建议在Mac OS X中使用此选项，因为 `getrlimit()` 函数返回一个不正确的值。

示例

```
<max_open_files>262144</max_open_files>
```

max_table_size_to_drop

限制删除表。

如果一个大小 **MergeTree** 表超过 `max_table_size_to_drop`（以字节为单位），您无法使用删除查询将其删除。

如果仍然需要在不重新启动ClickHouse服务器的情况下删除表，请创建 `<clickhouse-path>/flags/force_drop_table` 文件并运行DROP查询。

默认值：50GB。

值0表示您可以删除所有表而不受任何限制。

示例

```
<max_table_size_to_drop>0</max_table_size_to_drop>
```

merge_tree

微调中的表 MergeTree.

有关详细信息，请参阅MergeTreeSettings.h头文件。

示例

```
<merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</merge_tree>
```

openSSL

SSL客户端/服务器配置。

对SSL的支持由 libpoco 图书馆。该接口在文件中描述 [SSLManager.h](#)

服务器/客户端设置的密钥：

- privateKeyFile – The path to the file with the secret key of the PEM certificate. The file may contain a key and certificate at the same time.
- certificateFile – The path to the client/server certificate file in PEM format. You can omit it if privateKeyFile 包含证书。
- caConfig – The path to the file or directory that contains trusted root certificates. Details are in the description of the [A.背景](#) 同学们 可能的值: none, relaxed, strict, once.
- verificationMode – The method for checking the node's certificates. Details are in the description of the [A.背景](#) 同学们 可能的值: none, relaxed, strict, once.
- verificationDepth – The maximum length of the verification chain. Verification will fail if the certificate chain length exceeds the set value.
- loadDefaultCAFile – Indicates that built-in CA certificates for OpenSSL will be used. Acceptable values: true, false. |
- cipherList – Supported OpenSSL encryptions. For example: ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH.
- cacheSessions – Enables or disables caching sessions. Must be used in combination with sessionIdContext. 可接受的值: true, false.
- sessionIdContext – A unique set of random characters that the server appends to each generated identifier. The length of the string must not exceed `SSL_MAX_SSL_SESSION_ID_LENGTH`. 始终建议使用此参数，因为如果服务器缓存会话，以及客户端请求缓存，它有助于避免出现问题。默认值: \${application.name}.
- sessionCacheSize – The maximum number of sessions that the server caches. Default value: 1024*20. 0 – Unlimited sessions.
- sessionTimeout – Time for caching the session on the server.
- extendedVerification – Automatically extended verification of certificates after the session ends. Acceptable values: true, false.
- requireTLSv1 – Require a TLSv1 connection. Acceptable values: true, false.
- requireTLSv1_1 – Require a TLSv1.1 connection. Acceptable values: true, false.
- requireTLSv1_2 – Require a TLSv1.2 connection. Acceptable values: true, false.
- fips – Activates OpenSSL FIPS mode. Supported if the library's OpenSSL version supports FIPS.
- privateKeyPassphraseHandler – Class (PrivateKeyPassphraseHandler subclass) that requests the passphrase for accessing the private key. For example: <privateKeyPassphraseHandler>, <name>KeyFileHandler</name>, <options><password>test</password></options>, </privateKeyPassphraseHandler>.
- invalidCertificateHandler – Class (a subclass of CertificateHandler) for verifying invalid certificates. For example: <invalidCertificateHandler> <name>ConsoleCertificateHandler</name> </invalidCertificateHandler> .
- disableProtocols – Protocols that are not allowed to use.
- preferServerCiphers – Preferred server ciphers on the client.

```

<openSSL>
  <server>
    <!-- openssl req -subj "/CN=localhost" -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout /etc/clickhouse-server/server.key -out /etc/clickhouse-server/server.crt -->
    <certificateFile>/etc/clickhouse-server/server.crt</certificateFile>
    <privateKeyFile>/etc/clickhouse-server/server.key</privateKeyFile>
    <!-- openssl dhparam -out /etc/clickhouse-server/dhparam.pem 4096 -->
    <dhParamsFile>/etc/clickhouse-server/dhparam.pem</dhParamsFile>
    <verificationMode>none</verificationMode>
    <loadDefaultCAFile>true</loadDefaultCAFile>
    <cacheSessions>true</cacheSessions>
    <disableProtocols>sslv2,sslv3</disableProtocols>
    <preferServerCiphers>true</preferServerCiphers>
  </server>
  <client>
    <loadDefaultCAFile>true</loadDefaultCAFile>
    <cacheSessions>true</cacheSessions>
    <disableProtocols>sslv2,sslv3</disableProtocols>
    <preferServerCiphers>true</preferServerCiphers>
    <!-- Use for self-signed: <verificationMode>none</verificationMode> -->
    <invalidCertificateHandler>
      <!-- Use for self-signed: <name>AcceptCertificateHandler</name> -->
      <name>RejectCertificateHandler</name>
    </invalidCertificateHandler>
  </client>
</openSSL>

```

part_log

记录与之关联的事件 [MergeTree](#)。例如，添加或合并数据。您可以使用日志来模拟合并算法并比较它们的特征。您可以可视化合并过程。

查询记录在 [系统](#)。`part_log` 表，而不是在一个单独的文件。您可以在以下命令中配置此表的名称 `table` 参数（见下文）。

使用以下参数配置日志记录：

- `database` – Name of the database.
- `table` – Name of the system table.
- `partition_by` – Sets a [自定义分区键](#).
- `flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table.

示例

```

<part_log>
  <database>system</database>
  <table>part_log</table>
  <partition_by>toMonday(event_date)</partition_by>
  <flush_interval_milliseconds>7500</flush_interval_milliseconds>
</part_log>

```

路径

包含数据的目录的路径。

注

尾部斜杠是强制性的。

示例

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

query_log

用于记录接收到的查询的设置 **log_queries=1** 设置。

查询记录在 **系统**。**query_log** 表，而不是在一个单独的文件。您可以更改表的名称 **table** 参数（见下文）。

使用以下参数配置日志记录：

- **database** – Name of the database.
- **table** – Name of the system table the queries will be logged in.
- **partition_by** – Sets a **自定义分区键** 为了一张桌子
- **flush_interval_milliseconds** – Interval for flushing data from the buffer in memory to the table.

如果该表不存在，ClickHouse将创建它。如果在ClickHouse服务器更新时查询日志的结构发生了更改，则会重命名具有旧结构的表，并自动创建新表。

示例

```
<query_log>
<database>system</database>
<table>query_log</table>
<partition_by>toMonday(event_date)</partition_by>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
</query_log>
```

query_thread_log

设置用于记录接收到的查询的线程 **log_query_threads=1** 设置。

查询记录在 **系统**。**query_thread_log** 表，而不是在一个单独的文件。您可以更改表的名称 **table** 参数（见下文）。

使用以下参数配置日志记录：

- **database** – Name of the database.
- **table** – Name of the system table the queries will be logged in.
- **partition_by** – Sets a **自定义分区键** 对于一个系统表。
- **flush_interval_milliseconds** – Interval for flushing data from the buffer in memory to the table.

如果该表不存在，ClickHouse将创建它。如果更新ClickHouse服务器时查询线程日志的结构发生了更改，则会重命名具有旧结构的表，并自动创建新表。

示例

```
<query_thread_log>
<database>system</database>
<table>query_thread_log</table>
<partition_by>toMonday(event_date)</partition_by>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
</query_thread_log>
```

trace_log

设置为 **trace log** 系统表操作。

参数:

- `database` — Database for storing a table.
- `table` — Table name.
- `partition_by` — 自定义分区键 对于一个系统表。
- `flush_interval_milliseconds` — Interval for flushing data from the buffer in memory to the table.

默认服务器配置文件 `config.xml` 包含以下设置部分:

```
<trace_log>
  <database>system</database>
  <table>trace_log</table>
  <partition_by>toYYYYMM(event_date)</partition_by>
  <flush_interval_milliseconds>7500</flush_interval_milliseconds>
</trace_log>
```

query_masking_rules

基于正则表达式的规则，在将查询以及所有日志消息存储在服务器日志中之前，这些规则将应用于查询以及所有日志消息，`system.query_log`, `system.text_log`, `system.processes` 表，并在日志中发送给客户端。这允许防止从SQL查询敏感数据泄漏（如姓名，电子邮件，个人标识符或信用卡号码）记录。

示例

```
<query_masking_rules>
  <rule>
    <name>hide SSN</name>
    <regexp>(^|\D)\d{3}-\d{2}-\d{4}($|\D)</regexp>
    <replace>000-00-0000</replace>
  </rule>
</query_masking_rules>
```

配置字段:

- `name` - 规则的名称（可选）
- `regexp` - RE2兼容正则表达式（强制性）
- `replace` - 敏感数据的替换字符串（可选，默认情况下-六个星号）

屏蔽规则应用于整个查询（以防止敏感数据从格式错误/不可解析的查询泄漏）。

`system.events` 表有计数器 `QueryMaskingRulesMatch` 其中具有匹配的查询屏蔽规则的总数。

对于分布式查询，每个服务器必须单独配置，否则，子查询传递给其他节点将被存储而不屏蔽。

remote_servers

所使用的集群的配置 `分布` 表引擎和由 `cluster` 表功能。

示例

```
<remote_servers incl="clickhouse_remote_servers" />
```

对于该值 `incl` 属性，请参阅部分“[配置文件](#)”。

另请参阅

- [skip_unavailable_shards](#)

时区

服务器的时区。

指定为UTC时区或地理位置（例如，非洲/阿比让）的IANA标识符。

当DateTime字段输出为文本格式（打印在屏幕上或文件中）时，以及从字符串获取DateTime时，时区对于字符串和DateTime格式之间的转换是必需的。此外，如果在输入参数中没有收到时区，则时区用于处理时间和日期的函数。

示例

```
<timezone>Europe/Moscow</timezone>
```

tcp_port

通过TCP协议与客户端通信的端口。

示例

```
<tcp_port>9000</tcp_port>
```

tcp_port_secure

TCP端口，用于与客户端进行安全通信。使用它与 [OpenSSL](#) 设置。

可能的值

整数。

默认值

```
<tcp_port_secure>9440</tcp_port_secure>
```

mysql_port

通过MySQL协议与客户端通信的端口。

可能的值

整数。

示例

```
<mysql_port>9004</mysql_port>
```

tmp_path

用于处理大型查询的临时数据的路径。

注

尾部斜杠是强制性的。

示例

```
<tmp_path>/var/lib/clickhouse/tmp/</tmp_path>
```

tmp_policy

从政策 `storage_configuration` 存储临时文件。
如果没有设置 `tmp_path` 被使用，否则被忽略。

注

- `move_factor` 被忽略

- `keep_free_space_bytes` 被忽略
- `max_data_part_size_bytes` 被忽略
-您必须在该政策中只有一个卷

uncompressed_cache_size

表引擎使用的未压缩数据的缓存大小（以字节为单位）[MergeTree](#).

服务器有一个共享缓存。 内存按需分配。 如果选项使用缓存 `use_uncompressed_cache` 被启用。

在个别情况下，未压缩的缓存对于非常短的查询是有利的。

示例

```
<uncompressed_cache_size>8589934592</uncompressed_cache_size>
```

user_files_path

包含用户文件的目录。 在表函数中使用 [文件\(\)](#).

示例

```
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
```

users_config

包含文件的路径：

- 用户配置。
- 访问权限。
- 设置配置文件。
- 配额设置。

示例

```
<users_config>users.xml</users_config>
```

zookeeper

包含允许ClickHouse与 [zookpeer](#) 集群。

ClickHouse使用ZooKeeper存储复制表副本的元数据。 如果未使用复制的表，则可以省略此部分参数。

本节包含以下参数：

- `node` — ZooKeeper endpoint. You can set multiple endpoints.

例如：

```
<node index="1">
  <host>example_host</host>
  <port>2181</port>
</node>
```

The `index` attribute specifies the node order when trying to connect to the ZooKeeper cluster.

- `session_timeout` — Maximum timeout for the client session in milliseconds.
- `root` — The `znode` 隆隆隆路虜脢..陇.貌.垄拢卢虜祿.陇.貌路.隆拢脳枚脢虜.麓脢for脫 可选。
- `identity` — User and password, that can be required by ZooKeeper to give access to requested znodes. Optional.

配置示例

```
<zookeeper>
  <node>
    <host>example1</host>
    <port>2181</port>
  </node>
  <node>
    <host>example2</host>
    <port>2181</port>
  </node>
  <session_timeout_ms>30000</session_timeout_ms>
  <operation_timeout_ms>10000</operation_timeout_ms>
  <!-- Optional. Chroot suffix. Should exist. -->
  <root>/path/to/zookeeper/node</root>
  <!-- Optional. Zookeeper digest ACL string. -->
  <identity>user:password</identity>
</zookeeper>
```

另请参阅

- [复制](#)
- [动物园管理员程序员指南](#)

use_minimalistic_part_header_in_zookeeper

ZooKeeper中数据部分头的存储方法。

此设置仅适用于 `MergeTree` 家人 它可以指定：

- 在全局范围内 `merge_tree` 一节 `config.xml` 文件

ClickHouse使用服务器上所有表的设置。 您可以随时更改设置。 当设置更改时，现有表会更改其行为。

- 对于每个表。

创建表时，指定相应的 `发动机设置`。即使全局设置更改，具有此设置的现有表的行为也不会更改。

可能的值

- 0 — Functionality is turned off.
- 1 — Functionality is turned on.

如果 `use_minimalistic_part_header_in_zookeeper = 1`，然后 **复制** 表存储的数据部分的头紧凑使用一个单一的 `znode`。如果表包含许多列，则此存储方法显着减少了Zookeeper中存储的数据量。

注意

申请后 `use_minimalistic_part_header_in_zookeeper = 1`，您不能将ClickHouse服务器降级到不支持此设置的版本。在集群中的服务器上升级ClickHouse时要小心。不要一次升级所有服务器。在测试环境中或在集群的几台服务器上测试ClickHouse的新版本更安全。

Data part headers already stored with this setting can't be restored to their previous (non-compact) representation.

默认值: 0.

disable_internal_dns_cache

禁用内部DNS缓存。推荐用于在系统中运行ClickHouse随着频繁变化的基础设施，如Kubernetes。

默认值: 0.

dns_cache_update_period

更新存储在ClickHouse内部DNS缓存中的IP地址的周期（以秒为单位）。更新是在一个单独的系统线程中异步执行的。

默认值: 15.

服务器配置参数

本节包含无法在会话或查询级别更改的服务器设置的说明。

这些设置存储在 `config.xml` ClickHouse服务器上的文件。

Other settings are described in the «**设置**» section.

在研究设置之前，请阅读 **配置文件** 部分和注意使用替换（的 `incl` 和 `optional` 属性）。

查询权限

ClickHouse中的查询可以分为几种类型:

1. 读取数据查询: `SELECT`, `SHOW`, `DESCRIBE`, `EXISTS`.
2. 写入数据查询: `INSERT`, `OPTIMIZE`.
3. 更改设置查询: `SET`, `USE`.
4. **DDL** 查询: `CREATE`, `ALTER`, `RENAME`, `ATTACH`, `DETACH`, `DROP` `TRUNCATE`.
5. `KILL QUERY`.

以下设置按查询类型规范用户权限:

- **只读** — Restricts permissions for all types of queries except DDL queries.
- **allow_ddl** — Restricts permissions for DDL queries.

`KILL QUERY` 可以与任何设置进行。

只读

限制读取数据、写入数据和更改设置查询的权限。

查看查询如何划分为多种类型 [以上](#).

可能的值:

- 0 — All queries are allowed.
- 1 — Only read data queries are allowed.
- 2 — Read data and change settings queries are allowed.

设置后 `readonly = 1`，用户无法更改 `readonly` 和 `allow_ddl` 当前会话中的设置。

使用时 `GET` 方法中的 [HTTP接口](#)，`readonly = 1` 自动设置。要修改数据，请使用 `POST` 方法。

设置 `readonly = 1` 禁止用户更改所有设置。有一种方法可以禁止用户从只更改特定设置，有关详细信息，请参阅 [对设置的限制](#).

默认值: 0

allow_ddl

允许或拒绝 [DDL](#) 查询。

查看查询如何划分为多种类型 [以上](#).

可能的值:

- 0 — DDL queries are not allowed.
- 1 — DDL queries are allowed.

你不能执行 `SET allow_ddl = 1` 如果 `allow_ddl = 0` 对于当前会话。

默认值: 1

设置配置文件

设置配置文件是以相同名称分组的设置的集合。

信息

ClickHouse还支持 [SQL驱动的工作流](#) 用于管理设置配置文件。我们建议使用它。

配置文件可以有任何名称。配置文件可以有任何名称。您可以为不同的用户指定相同的配置文件。您可以在设置配置文件中编写最重要的事情是 `readonly=1`，这确保只读访问。

设置配置文件可以彼此继承。要使用继承，请指示一个或多个 `profile` 配置文件中列出的其他设置之前的设置。如果在不同的配置文件中定义了一个设置，则使用最新定义。

要应用配置文件中的所有设置，请设置 `profile` 设置。

示例:

安装 `web` 侧写

```
SET profile = 'web'
```

设置配置文件在用户配置文件中声明。这通常是 `users.xml`.

示例:

```

<!-- Settings profiles -->
<profiles>
    <!-- Default settings -->
    <default>
        <!-- The maximum number of threads when running a single query. -->
        <max_threads>8</max_threads>
    </default>

    <!-- Settings for queries from the user interface -->
    <web>
        <max_rows_to_read>1000000000</max_rows_to_read>
        <max_bytes_to_read>1000000000000</max_bytes_to_read>

        <max_rows_to_group_by>1000000</max_rows_to_group_by>
        <group_by_overflow_mode>any</group_by_overflow_mode>

        <max_rows_to_sort>1000000</max_rows_to_sort>
        <max_bytes_to_sort>10000000000</max_bytes_to_sort>

        <max_result_rows>100000</max_result_rows>
        <max_result_bytes>1000000000</max_result_bytes>
        <result_overflow_mode>break</result_overflow_mode>

        <max_execution_time>600</max_execution_time>
        <min_execution_speed>1000000</min_execution_speed>
        <timeout_before_checking_execution_speed>15</timeout_before_checking_execution_speed>

        <max_columns_to_read>25</max_columns_to_read>
        <max_temporary_columns>100</max_temporary_columns>
        <max_temporary_non_const_columns>50</max_temporary_non_const_columns>

        <max_subquery_depth>2</max_subquery_depth>
        <max_pipeline_depth>25</max_pipeline_depth>
        <max_ast_depth>50</max_ast_depth>
        <max_ast_elements>100</max_ast_elements>

        <readonly>1</readonly>
    </web>
</profiles>

```

该示例指定了两个配置文件: `default` 和 `web`.

该 `default` 配置文件有一个特殊用途：它必须始终存在并在启动服务器时应用。换句话说，`default` 配置文件包含默认设置。

该 `web` 配置文件是一个常规的配置文件，可以使用设置 `SET` 查询或在HTTP查询中使用URL参数。

对设置的限制

在设置的约束可以在定义 `profiles` 一节 `user.xml` 配置文件，并禁止用户更改一些设置与 `SET` 查询。

约束定义如下：

```

<profiles>
    <user_name>
        <constraints>
            <setting_name_1>
                <min>lower boundary</min>

```

```

</setting_name_1>
<setting_name_2>
  <max>upper_boundary</max>
</setting_name_2>
<setting_name_3>
  <min>lower_boundary</min>
  <max>upper_boundary</max>
</setting_name_3>
<setting_name_4>
  <readonly/>
</setting_name_4>
</constraints>
</user_name>
</profiles>

```

如果用户试图违反约束，将引发异常，并且设置不会更改。

支持三种类型的约束: `min`, `max`, `readonly`. 该 `min` 和 `max` 约束指定数值设置的上边界和下边界，并且可以组合使用。该 `readonly` constraint 指定用户根本无法更改相应的设置。

示例：让 `users.xml` 包括行:

```

<profiles>
  <default>
    <max_memory_usage>10000000000</max_memory_usage>
    <force_index_by_date>0</force_index_by_date>
    ...
    <constraints>
      <max_memory_usage>
        <min>5000000000</min>
        <max>20000000000</max>
      </max_memory_usage>
      <force_index_by_date>
        <readonly/>
      </force_index_by_date>
    </constraints>
  </default>
</profiles>

```

以下查询都会引发异常:

```

SET max_memory_usage=20000000001;
SET max_memory_usage=4999999999;
SET force_index_by_date=1;

```

```

Code: 452, e.displayText() = DB::Exception: Setting max_memory_usage should not be greater than 20000000000.
Code: 452, e.displayText() = DB::Exception: Setting max_memory_usage should not be less than 50000000000.
Code: 452, e.displayText() = DB::Exception: Setting force_index_by_date should not be changed.

```

注：该 `default` 配置文件具有特殊的处理：所有定义的约束 `default` 配置文件成为默认约束，因此它们限制所有用户，直到为这些用户显式覆盖它们。

用户设置

该 `users` 一节 `user.xml` 配置文件包含用户设置。

信息

ClickHouse还支持 **SQL**驱动的工作流 用于管理用户。 我们建议使用它。

的结构 `users` 科:

```
<users>
  <!-- If user name was not specified, 'default' user is used. -->
  <user_name>
    <password></password>
    <!-- Or -->
    <password_sha256_hex></password_sha256_hex>

    <access_management>0|1</access_management>

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

    <profile>profile_name</profile>

    <quota>default</quota>

    <databases>
      <database_name>
        <table_name>
          <filter>expression</filter>
        <table_name>
      </database_name>
    </databases>
  </user_name>
  <!-- Other users settings -->
</users>
```

用户名/密码

密码可以以明文或**SHA256**（十六进制格式）指定。

- 以明文形式分配密码（不推荐），把它放在一个 `password` 元素。

例如，`<password>qwerty</password>`. 密码可以留空。

- 要使用其**SHA256**散列分配密码，请将其放置在 `password_sha256_hex` 元素。

例如，

```
<password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sh
a256_hex>.
```

如何从shell生成密码的示例：

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

结果的第一行是密码。 第二行是相应的**SHA256**哈希。

- 为了与MySQL客户端兼容，密码可以在双**SHA1**哈希中指定。 放进去 `password_double_sha1_hex` 元素。

例如，

```
<password_double_sha1_hex>08b4a0f1de6ad37da17359e592c8d74788a83eb0</password_double_sha1_hex>.
```

如何从shell生成密码的示例:

```
PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | shasum | tr -d '-' | xxd -r -p | shasum | tr -d '-'
```

结果的第一行是密码。第二行是相应的双SHA1哈希。

访问管理

此设置启用禁用使用SQL驱动 [访问控制和帐户管理](#) 对于用户。

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值: 0。

用户名/网络

用户可以从中连接到ClickHouse服务器的网络列表。

列表中的每个元素都可以具有以下形式之一:

- `<ip>` — IP address or network mask.

例: 213.180.204.3, 10.0.0.1/8, 10.0.0.1/255.255.255.0, 2a02:6b8::3, 2a02:6b8::3/64, 2a02:6b8::3/ffff:ffff:ffff:ffff::.

- `<host>` — Hostname.

示例: example01.host.ru.

要检查访问，将执行DNS查询，并将所有返回的IP地址与对等地址进行比较。

- `<host_regex>` — Regular expression for hostnames.

示例, ^example\d\d-\d\d\d\.\host\.ru\$

要检查访问，a [DNS PTR查询](#) 对对等体地址执行，然后应用指定的正则表达式。然后，对PTR查询的结果执行另一个DNS查询，并将所有接收到的地址与对等地址进行比较。我们强烈建议正则表达式以\$结尾。

DNS请求的所有结果都将被缓存，直到服务器重新启动。

例

要从任何网络打开用户的访问权限，请指定:

```
<ip>>::/0</ip>
```

警告

从任何网络开放访问是不安全的，除非你有一个防火墙正确配置或服务器没有直接连接到互联网。

若要仅从本地主机打开访问权限，请指定:

```
<ip>>::1</ip>
<ip>127.0.0.1</ip>
```

user_name/profile

您可以为用户分配设置配置文件。设置配置文件在单独的部分配置 `users.xml` 文件有关详细信息，请参阅 [设置配置文件](#)。

用户名/配额

配额允许您在一段时间内跟踪或限制资源使用情况。配额在配置 `quotas` 一节 `users.xml` 配置文件。

您可以为用户分配配额。有关配额配置的详细说明，请参阅 [配额](#)。

用户名/数据库

在本节中，您可以限制ClickHouse返回的行 `SELECT` 由当前用户进行的查询，从而实现基本的行级安全性。

示例

以下配置强制该用户 `user1` 只能看到的行 `table1` 作为结果 `SELECT` 查询，其中的值 `id` 场是1000。

```
<user1>
  <databases>
    <database_name>
      <table1>
        <filter>id = 1000</filter>
      </table1>
    </database_name>
  </databases>
</user1>
```

该 `filter` 可以是导致任何表达式 `UInt8`-键入值。它通常包含比较和逻辑运算符。从行 `database_name.table1` 其中，不会为此用户返回为0的筛选结果。过滤是不兼容的 `PREWHERE` 操作和禁用 `WHERE→PREWHERE` 优化。

查询复杂性的限制

对查询复杂性的限制是设置的一部分。

它们被用来从用户界面提供更安全的执行。

几乎所有的限制只适用于选择。对于分布式查询处理，每个服务器上分别应用限制。

Restrictions on the «maximum amount of something» can take the value 0, which means «unrestricted».

大多数限制也有一个 ‘overflow_mode’ 设置，这意味着超过限制时该怎么做。

它可以采用以下两个值之一: `throw` 或 `break`. 对聚合的限制(`group_by_overflow_mode`)也具有以下值 `any`.

`throw` – Throw an exception (default).

`break` – Stop executing the query and return the partial result, as if the source data ran out.

`any` (only for `group_by_overflow_mode`) – Continuing aggregation for the keys that got into the set, but don't add new keys to the set.

只读

值为0时，可以执行任何查询。

如果值为1，则只能执行读取请求（如`SELECT`和`SHOW`）。禁止写入和更改设置（插入，设置）的请求。

值为2时，可以处理读取查询（选择、显示）和更改设置（设置）。

启用只读模式后，您无法在当前会话中禁用它。

在HTTP接口中使用GET方法时，‘`readonly = 1`’ 自动设置。换句话说，对于修改数据的查询，您只能使用POST方法。您可以在POST正文或URL参数中发送查询本身。

max memory usage

max_memory_usage

用于在单个服务器上运行查询的最大RAM量。

在默认配置文件中，最大值为10GB。

该设置不考虑计算机上的可用内存量或内存总量。

该限制适用于单个服务器中的单个查询。

您可以使用 `SHOW PROCESSLIST` 查看每个查询的当前内存消耗。

此外，还会跟踪每个查询的内存消耗峰值并将其写入日志。

不监视某些聚合函数的状态的内存使用情况。

未完全跟踪聚合函数的状态的内存使用情况 `min`, `max`, `any`, `anyLast`, `argMin`, `argMax` 从 `String` 和 `Array` 争论。

内存消耗也受到参数的限制 `max_memory_usage_for_user` 和 `max_memory_usage_for_all_queries`.

max_memory_usage_for_user

用于在单个服务器上运行用户查询的最大RAM量。

默认值定义在 [设置。h](#). 默认情况下，金额不受限制 (`max_memory_usage_for_user = 0`).

另请参阅说明 [max_memory_usage](#).

max_memory_usage_for_all_queries

用于在单个服务器上运行所有查询的最大RAM数量。

默认值定义在 [设置。h](#). 默认情况下，金额不受限制 (`max_memory_usage_for_all_queries = 0`).

另请参阅说明 [max_memory_usage](#).

max_rows_to_read

可以在每个块（而不是每行）上检查以下限制。也就是说，限制可以打破一点。

在多个线程中运行查询时，以下限制单独应用于每个线程。

运行查询时可从表中读取的最大行数。

max_bytes_to_read

运行查询时可以从表中读取的最大字节数（未压缩数据）。

read_overflow_mode

读取的数据量超过其中一个限制时该怎么办: ‘throw’ 或 ‘break’. 默认情况下，扔。

max_rows_to_group_by

从聚合接收的唯一密钥的最大数量。此设置允许您在聚合时限制内存消耗。

group_by_overflow_mode

当聚合的唯一键数超过限制时该怎么办: ‘throw’, ‘break’，或 ‘any’. 默认情况下，扔。

使用 ‘any’ 值允许您运行GROUP BY的近似值。这种近似值的质量取决于数据的统计性质。

max_rows_to_sort

排序前的最大行数。这允许您在排序时限制内存消耗。

max_bytes_to_sort

排序前的最大字节数。

sort_overflow_mode

如果排序前收到的行数超过其中一个限制，该怎么办: ‘throw’ 或 ‘break’. 默认情况下，扔。

max_result_rows

限制结果中的行数。还检查子查询，并在运行分布式查询的部分时在远程服务器上。

max_result_bytes

限制结果中的字节数。与之前的设置相同。

result_overflow_mode

如果结果的体积超过其中一个限制，该怎么办: ‘throw’ 或 ‘break’. 默认情况下，扔。

使用 ‘break’ 类似于使用限制。

max_execution_time

最大查询执行时间（以秒为单位）。

此时，不会检查其中一个排序阶段，也不会在合并和最终确定聚合函数时进行检查。

timeout_overflow_mode

如果查询的运行时间长于 ‘max_execution_time’: ‘throw’ 或 ‘break’. 默认情况下，扔。

min_execution_speed

以每秒行为单位的最小执行速度。检查每个数据块时 ‘timeout_before_checking_execution_speed’ 到期。如果执行速度较低，则会引发异常。

timeout_before_checking_execution_speed

检查执行速度是不是太慢（不低于 ‘min_execution_speed’），在指定的时间以秒为单位已过期之后。

max_columns_to_read

单个查询中可从表中读取的最大列数。如果查询需要读取更多列，则会引发异常。

max_temporary_columns

运行查询时必须同时保留在RAM中的最大临时列数，包括常量列。如果有比这更多的临时列，它会引发异常。

max_temporary_non_const_columns

同样的事情 ‘max_temporary_columns’，但不计数常数列。

请注意，常量列在运行查询时经常形成，但它们需要大约零计算资源。

max_subquery_depth

子查询的最大嵌套深度。如果子查询更深，则会引发异常。默认情况下，100。

max_pipeline_depth

最大管道深度。对应于查询处理期间每个数据块经历的转换数。在单个服务器的限制范围内计算。如果管道深度较大，则会引发异常。默认情况下，1000。

max_ast_depth

查询语法树的最大嵌套深度。如果超出，将引发异常。

此时，在解析过程中不会对其进行检查，而是仅在解析查询之后进行检查。也就是说，在分析过程中可以创建一个太深的语法树，但查询将失败。默认情况下，1000。

max_ast_elements

查询语法树中的最大元素数。如果超出，将引发异常。

与前面的设置相同，只有在解析查询后才会检查它。默认情况下，50,000。

max_rows_in_set

从子查询创建的IN子句中数据集的最大行数。

max_bytes_in_set

从子查询创建的IN子句中的集合使用的最大字节数（未压缩数据）。

set_overflow_mode

当数据量超过其中一个限制时该怎么办：'throw' 或 'break'。默认情况下，扔。

max_rows_in_distinct

使用DISTINCT时的最大不同行数。

max_bytes_in_distinct

使用DISTINCT时哈希表使用的最大字节数。

distinct_overflow_mode

当数据量超过其中一个限制时该怎么办：'throw' 或 'break'。默认情况下，扔。

max_rows_to_transfer

使用GLOBAL IN时，可以传递到远程服务器或保存在临时表中的最大行数。

max_bytes_to_transfer

使用GLOBAL IN时，可以传递到远程服务器或保存在临时表中的最大字节数（未压缩数据）。

transfer_overflow_mode

当数据量超过其中一个限制时该怎么办：'throw' 或 'break'。默认情况下，扔。

max_rows_in_join

Limits the number of rows in the hash table that is used when joining tables.

This setting applies to **SELECT ... JOIN** operations and the **Join** table engine.

If a query contains multiple joins, ClickHouse checks this setting for every intermediate result.

ClickHouse can proceed with different actions when the limit is reached. Use the **join_overflow_mode** setting to choose the action.

Possible values:

- Positive integer.
- 0 — Unlimited number of rows.

Default value: 0.

max_bytes_in_join

Limits the size in bytes of the hash table used when joining tables.

This setting applies to **SELECT ... JOIN** operations and **Join** table engine.

If the query contains joins, ClickHouse checks this setting for every intermediate result.

ClickHouse can proceed with different actions when the limit is reached. Use `join_overflow_mode` settings to choose the action.

Possible values:

- Positive integer.
- 0 — Memory control is disabled.

Default value: 0.

join_overflow_mode

Defines what action ClickHouse performs when any of the following join limits is reached:

- `max_bytes_in_join`
- `max_rows_in_join`

Possible values:

- `THROW` — ClickHouse throws an exception and breaks operation.
- `BREAK` — ClickHouse breaks operation and doesn't throw an exception.

Default value: `THROW`.

See Also

- [JOIN clause](#)
- [Join table engine](#)

max_bytes_before_external_group_by

Enables or disables execution of `GROUP BY` clauses in external memory. See [GROUP BY in external memory](#).

Possible values:

- Maximum volume of RAM (in bytes) that can be used by the single `GROUP BY` operation.
- 0 — `GROUP BY` in external memory disabled.

Default value: 0.

设置

分布_产品_模式

改变的行为 [分布式子查询](#).

ClickHouse applies this setting when the query contains the product of distributed tables, i.e. when the query for a distributed table contains a non-GLOBAL subquery for the distributed table.

限制:

- 仅适用于IN和JOIN子查询。
- 仅当FROM部分使用包含多个分片的分布式表时。
- 如果子查询涉及包含多个分片的分布式表。
- 不用于表值[远程](#)功能。

可能的值:

- `deny` — Default value. Prohibits using these types of subqueries (returns the “Double-distributed in/JOIN subqueries is denied” 例外)。
- `local` — Replaces the database and table in the subquery with local ones for the destination server (shard), leaving the normal `IN/JOIN`.
- `global` — Replaces the `IN/JOIN` 查询与 `GLOBAL IN/GLOBAL JOIN`.
- `allow` — Allows the use of these types of subqueries.

enable_optimize_predicate_expression

打开谓词下推 `SELECT` 查询。

谓词下推可以显着减少分布式查询的网络流量。

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 1。

用途

请考虑以下查询:

1. `SELECT count() FROM test_table WHERE date = '2018-10-10'`
2. `SELECT count() FROM (SELECT * FROM test_table) WHERE date = '2018-10-10'`

如果 `enable_optimize_predicate_expression = 1`，则这些查询的执行时间相等，因为 ClickHouse 应用 `WHERE` 对子查询进行处理。

如果 `enable_optimize_predicate_expression = 0`，那么第二个查询的执行时间要长得多，因为 `WHERE` 子句适用于子查询完成后的所有数据。

fallback_to_stale_replicas_for_distributed_queries

如果更新的数据不可用，则强制对过期副本进行查询。看 [复制](#)。

ClickHouse 从表的过时副本中选择最相关的副本。

执行时使用 `SELECT` 从指向复制表的分布式表。

默认情况下，1 (已启用)。

force_index_by_date

如果索引不能按日期使用，则禁用查询执行。

适用于 MergeTree 系列中的表。

如果 `force_index_by_date=1`，ClickHouse 检查查询是否具有可用于限制数据范围的 `date` 键条件。如果没有合适的条件，则会引发异常。但是，它不检查条件是否减少了要读取的数据量。例如，条件 `Date != '2000-01-01'` 即使它与表中的所有数据匹配（即运行查询需要完全扫描），也是可以接受的。有关 MergeTree 表中数据范围的详细信息，请参阅 [MergeTree](#)。

force_primary_key

如果无法按主键编制索引，则禁用查询执行。

适用于 MergeTree 系列中的表。

如果 `force_primary_key=1`，ClickHouse 检查查询是否具有可用于限制数据范围的主键条件。如果没有合适的条件，则会引发异常。但是，它不检查条件是否减少了要读取的数据量。有关 MergeTree 表中数据范围的详细信息，请参阅 [MergeTree](#)。

format schema

initial_schema

当您使用需要架构定义的格式时，此参数非常有用，例如 [普罗托船长](#) 或 [Protobuf](#). 该值取决于格式。

fsync_metadata

启用或禁用 [fsync](#) 写作时 `.sql` 文件 默认情况下启用。

如果服务器有数百万个不断创建和销毁的小表，那么禁用它是有意义的。

enable_http_compression

在对HTTP请求的响应中启用或禁用数据压缩。

欲了解更多信息，请阅读 [HTTP接口描述](#).

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 0 。

http_zlib_compression_level

在以下情况下，设置对HTTP请求的响应中的数据压缩级别 [enable_http_compression=1](#).

可能的值：数字从1到9。

默认值 : 3 。

http_native_compression_disable_checksumming_on_decompression

在从客户端解压缩HTTP POST数据时启用或禁用校验和验证。仅用于ClickHouse原生压缩格式（不用于 `gzip` 或 `deflate`）。

欲了解更多信息，请阅读 [HTTP接口描述](#).

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 0 。

send_progress_in_http_headers

启用或禁用 `X-ClickHouse-Progress` Http响应头 `clickhouse-server` 答复。

欲了解更多信息，请阅读 [HTTP接口描述](#).

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 0 。

max_http_get_redirects

限制HTTP GET重定向跳数的最大数量 [URL-发动机](#) 表。该设置适用于两种类型的表：由 [CREATE TABLE](#) 查询和由 `url` 表功能。

可能的值:

- 跳数的任何正整数。

- 0 — No hops allowed.

默认值：0。

input_format_allow_errors_num

设置从文本格式（CSV，TSV等）读取时可接受的错误的最大数量。).

默认值为0。

总是与它配对 `input_format_allow_errors_ratio`.

如果在读取行时发生错误，但错误计数器仍小于 `input_format_allow_errors_num`，ClickHouse忽略该行并移动到下一个。

如果两者 `input_format_allow_errors_num` 和 `input_format_allow_errors_ratio` 超出时，ClickHouse引发异常。

input_format_allow_errors_ratio

设置从文本格式（CSV，TSV等）读取时允许的最大错误百分比。).

错误百分比设置为介于0和1之间的浮点数。

默认值为0。

总是与它配对 `input_format_allow_errors_num`.

如果在读取行时发生错误，但错误计数器仍小于 `input_format_allow_errors_ratio`，ClickHouse忽略该行并移动到下一个。

如果两者 `input_format_allow_errors_num` 和 `input_format_allow_errors_ratio` 超出时，ClickHouse引发异常。

input_format_values_interpret_expressions

如果快速流解析器无法解析数据，则启用或禁用完整SQL解析器。此设置仅用于 **值** 格式在数据插入。有关语法分析的详细信息，请参阅 [语法](#) 科。

可能的值:

- 0 — Disabled.

在这种情况下，您必须提供格式化的数据。见 [格式](#) 科。

- 1 — Enabled.

在这种情况下，您可以使用SQL表达式作为值，但数据插入速度要慢得多。如果仅插入格式化的数据，则ClickHouse的行为就好像设置值为0。

默认值：1。

使用示例

插入 [日期时间](#) 使用不同的设置键入值。

```
SET input_format_values_interpret_expressions = 0;
INSERT INTO datetime_t VALUES (now())
```

```
Exception on client:
Code: 27. DB::Exception: Cannot parse input: expected ) before: now(): (at row 1)
```

```
SET input_format_values_interpret_expressions = 1;
INSERT INTO datetime_t VALUES (now())
```

Ok.

最后一个查询等效于以下内容:

```
SET input_format_values_interpret_expressions = 0;
INSERT INTO datetime_t SELECT now()
```

Ok.

input_format_values_deduce_templates_of_expressions

启用或禁用以下内容中的SQL表达式的模板扣除 值 格式。它允许解析和解释表达式 Values 如果连续行中的表达式具有相同的结构，速度要快得多。ClickHouse尝试推导表达式的模板，使用此模板解析以下行，并在一批成功解析的行上评估表达式。

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值: 1。

对于以下查询:

```
INSERT INTO test VALUES (lower('Hello')), (lower('world')), (lower('INSERT')), (upper('Values')), ...
```

- 如果 `input_format_values_interpret_expressions=1` 和 `format_values_deduce_templates_of_expressions=0`，表达式为每行分别解释（对于大量行来说，这非常慢）。
- 如果 `input_format_values_interpret_expressions=0` 和 `format_values_deduce_templates_of_expressions=1`，第一，第二和第三行中的表达式使用template解析 `lower(String)` 并一起解释，第四行中的表达式用另一个模板解析 `(upper(String))`。
- 如果 `input_format_values_interpret_expressions=1` 和 `format_values_deduce_templates_of_expressions=1`，与前面的情况相同，但如果不可能推导出模板，也允许回退到单独解释表达式。

input_format_values_accurate_types_of_literals

此设置仅在以下情况下使用 `input_format_values_deduce_templates_of_expressions = 1`。它可能发生，某些列的表达式具有相同的结构，但包含不同类型的数字文字，例如

```
(..., abs(0), ...),          -- UInt64 literal
(..., abs(3.141592654), ...), -- Float64 literal
(..., abs(-1), ...),         -- Int64 literal
```

可能的值:

- 0 — Disabled.

In this case, ClickHouse may use a more general type for some literals (e.g., `Float64` 或 `Int64` 而不是 `UInt64` 为 42)，但它可能会导致溢出和精度问题。

- 1 — Enabled.

在这种情况下，ClickHouse会检查文本的实际类型，并使用相应类型的表达式模板。在某些情况下，可能会显着减慢表达式评估 Values.

默认值：1。

input_format_defaults_for_omitted_fields

执行时 `INSERT` 查询时，将省略的输入列值替换为相应列的默认值。此选项仅适用于 `JSONEachRow`, `CSV` 和 `TabSeparated` 格式。

注

启用此选项后，扩展表元数据将从服务器发送到客户端。它会消耗服务器上的额外计算资源，并可能降低性能。

可能的值：

- 0 — Disabled.
- 1 — Enabled.

默认值：1。

input_format_tsv_empty_as_default

启用后，将TSV中的空输入字段替换为默认值。对于复杂的默认表达式 `input_format_defaults_for_omitted_fields` 必须启用了。

默认情况下禁用。

input_format_null_as_default

如果输入数据包含 `NULL`，但相应列的数据类型不 `Nullable(T)`（对于文本输入格式）。

input_format_skip_unknown_fields

启用或禁用跳过额外数据的插入。

写入数据时，如果输入数据包含目标表中不存在的列，ClickHouse将引发异常。如果启用了跳过，ClickHouse不会插入额外的数据，也不会引发异常。

支持的格式：

- `JSONEachRow`
- `CSVWithNames`
- `TabSeparatedWithNames`
- `TSKV`

可能的值：

- 0 — Disabled.
- 1 — Enabled.

默认值：0。

input_format_import_nested_json

启用或禁用具有嵌套对象的JSON数据的插入。

支持的格式：

- `JSONEachRow`

可能的值：

- 0 — Disabled.

- 1 — Enabled.

默认值：0。

另请参阅：

- 嵌套结构的使用 与 `JSONEachRow` 格式。

input_format_with_names_use_header

启用或禁用插入数据时检查列顺序。

为了提高插入性能，如果您确定输入数据的列顺序与目标表中的列顺序相同，建议禁用此检查。

支持的格式：

- `CSVWithNames`
- `TabSeparatedWithNames`

可能的值：

- 0 — Disabled.
- 1 — Enabled.

默认值：1。

date_time_input_format

允许选择日期和时间的文本表示的解析器。

该设置不适用于 [日期和时间功能](#).

可能的值：

- `'best_effort'` — Enables extended parsing.

ClickHouse可以解析基本 `YYYY-MM-DD HH:MM:SS` 格式和所有 [ISO 8601](#) 日期和时间格式。例如, `'2018-06-08T01:02:03.000Z'`.

- `'basic'` — Use basic parser.

ClickHouse只能解析基本的 `YYYY-MM-DD HH:MM:SS` 格式。例如, `'2019-08-20 10:18:56'`.

默认值: `'basic'`.

另请参阅：

- [日期时间数据类型](#)。
- [用于处理日期和时间的函数](#)。

join_default_strictness

设置默认严格性 [加入子句](#).

可能的值：

- `ALL` — If the right table has several matching rows, ClickHouse creates a [笛卡尔积](#) 从匹配的行。这是正常的 `JOIN` 来自标准SQL的行为。
- `ANY` — If the right table has several matching rows, only the first one found is joined. If the right table has only one matching row, the results of `ANY` 和 `ALL` 都是一样的
- `ASOF` — For joining sequences with an uncertain match.
- `Empty string` — If `ALL` 或 `ANY` 如果未在查询中指定，ClickHouse将引发异常。

默认值: `ALL`

join_any_take_last_row

更改联接操作的行为 ANY 严格。

注意

此设置仅适用于 JOIN 操作与 加入我们 发动机表.

可能的值:

- 0 — If the right table has more than one matching row, only the first one found is joined.
- 1 — If the right table has more than one matching row, only the last one found is joined.

默认值 : 0。

另请参阅:

- [JOIN子句](#)
- [联接表引擎](#)
- [join_default_strictness](#)

join_use_nulls

设置类型 JOIN 行为 合并表时，可能会出现空单元格。 ClickHouse根据此设置以不同的方式填充它们。

可能的值:

- 0 — The empty cells are filled with the default value of the corresponding field type.
- 1 — JOIN 其行为方式与标准SQL中的行为方式相同。 相应字段的类型将转换为 可为空，和空单元格填充 NULL.

默认值 : 0。

max_block_size

在ClickHouse中，数据由块（列部分集）处理。 单个块的内部处理周期足够高效，但每个块都有明显的支出。 该 max_block_size 设置是建议从表中加载块的大小（行数）。 块大小不应该太小，以便每个块上的支出仍然明显，但不能太大，以便在第一个块处理完成后快速完成限制查询。 目标是避免在多个线程中提取大量列时占用太多内存，并且至少保留一些缓存局部性。

默认值 : 65,536。

块的大小 max_block_size 并不总是从表中加载。 如果显然需要检索的数据较少，则处理较小的块。

preferred_block_size_bytes

用于相同的目的 max_block_size，但它通过使其适应块中的行数来设置推荐的块大小（以字节为单位）。

但是，块大小不能超过 max_block_size 行。

默认情况下 : 1,000,000。 它只有在从MergeTree引擎读取时才有效。

merge_tree_min_rows_for_concurrent_read

如果从a的文件中读取的行数 MergeTree 表超过 merge_tree_min_rows_for_concurrent_read 然后ClickHouse尝试在多个线程上从该文件执行并发读取。

可能的值:

- 任何正整数。

默认值:163840.

`merge_tree_min_bytes_for_concurrent_read`

如果从一个文件中读取的字节数 MergeTree-发动机表超过 `merge_tree_min_bytes_for_concurrent_read`，然后ClickHouse尝试在多个线程中并发读取此文件。

可能的值:

- 任何正整数。

默认值:251658240.

`merge_tree_min_rows_for_seek`

如果要在文件中读取的两个数据块之间的距离小于 `merge_tree_min_rows_for_seek` 行，然后ClickHouse不查找文件，而是按顺序读取数据。

可能的值:

- 任何正整数。

默认值：0。

`merge_tree_min_bytes_for_seek`

如果要在文件中读取的两个数据块之间的距离小于 `merge_tree_min_bytes_for_seek` 字节数，然后ClickHouse依次读取包含两个块的文件范围，从而避免了额外的寻道。

可能的值:

- 任何正整数。

默认值：0。

`merge_tree_coarse_index_granularity`

搜索数据时，ClickHouse会检查索引文件中的数据标记。如果ClickHouse发现所需的键在某个范围内，它将此范围划分为 `merge_tree_coarse_index_granularity` 子范围和递归地搜索所需的键。

可能的值:

- 任何正偶数整数。

默认值：8。

`merge_tree_max_rows_to_use_cache`

如果克里克豪斯应该阅读更多 `merge_tree_max_rows_to_use_cache` 在一个查询中的行，它不使用未压缩块的缓存。

未压缩块的缓存存储为查询提取的数据。ClickHouse使用此缓存来加快对重复的小查询的响应。此设置可保护缓存免受读取大量数据的查询的破坏。该 `uncompressed_cache_size` 服务器设置定义未压缩块的高速缓存的大小。

可能的值:

- 任何正整数。

Default value: 128 × 8192.

`merge_tree_max_bytes_to_use_cache`

如果克里克豪斯应该阅读更多 `merge_tree_max_bytes_to_use_cache` 在一个查询中的字节，它不使用未压缩块的缓存。

未压缩块的缓存存储为查询提取的数据。ClickHouse使用此缓存来加快对重复的小查询的响应。此设置可保护缓存免受读取大量数据的查询的破坏。该 `uncompressed_cache_size` 服务器设置定义未压缩块的高速缓存的大小。

可能的值:

- 任何正整数。

默认值: 2013265920.

min_bytes_to_use_direct_io

使用直接I/O访问存储磁盘所需的最小数据量。

ClickHouse在从表中读取数据时使用此设置。如果要读取的所有数据的总存储量超过 `min_bytes_to_use_direct_io` 字节，然后ClickHouse读取从存储磁盘的数据 `O_DIRECT` 选项。

可能的值:

- 0 — Direct I/O is disabled.
- 整数。

默认值: 0。

log_queries

设置查询日志记录。

使用此设置发送到ClickHouse的查询将根据以下内容中的规则记录 `query_log` 服务器配置参数。

示例:

```
log_queries=1
```

log_queries_min_type

`query_log` 要记录的最小类型。

可能的值:

- `QUERY_START` (=1)
- `QUERY_FINISH` (=2)
- `EXCEPTION_BEFORE_START` (=3)
- `EXCEPTION_WHILE_PROCESSING` (=4)

默认值: `QUERY_START`.

可以用来限制哪些entiries将去 `query_log`，说你只有在错误中才感兴趣，那么你可以使用 `EXCEPTION_WHILE_PROCESSING`:

```
log_queries_min_type='EXCEPTION_WHILE_PROCESSING'
```

log_query_threads

设置查询线程日志记录。

ClickHouse使用此设置运行的查询线程将根据以下命令中的规则记录 `query_thread_log` 服务器配置参数。

示例:

```
log_query_threads=1
```

max_insert_block_size

要插入到表中的块的大小。

此设置仅适用于服务器形成块的情况。

例如，对于通过HTTP接口进行的插入，服务器会分析数据格式并形成指定大小的块。

但是当使用clickhouse-client时，客户端解析数据本身，并且‘max_insert_block_size’服务器上的设置不会影响插入的块的大小。

使用INSERT SELECT时，该设置也没有目的，因为数据是使用在SELECT之后形成的相同块插入的。

默认值：1,048,576。

默认值略高于 max_block_size。这样做的原因是某些表引擎 (*MergeTree) 在磁盘上为每个插入的块形成一个数据部分，这是一个相当大的实体。同样，*MergeTree 表在插入过程中对数据进行排序，并且足够大的块大小允许在RAM中对更多数据进行排序。

min_insert_block_size_rows

设置块中可以通过以下方式插入到表中的最小行数 INSERT 查询。较小尺寸的块被压扁成较大的块。

可能的值：

- 整数。
- 0 — Squashing disabled.

默认值：1048576。

min_insert_block_size_bytes

设置块中的最小字节数，可以通过以下方式插入到表中 INSERT 查询。较小尺寸的块被压扁成较大的块。

可能的值：

- 整数。
- 0 — Squashing disabled.

默认值：268435456。

max_replica_delay_for_distributed_queries

禁用分布式查询的滞后副本。看 [复制](#)。

以秒为单位设置时间。如果副本滞后超过设定值，则不使用此副本。

默认值：300。

执行时使用 SELECT 从指向复制表的分布式表。

max_threads

查询处理线程的最大数量，不包括用于从远程服务器检索数据的线程（请参阅‘max_distributed_connections’参数）。

此参数适用于并行执行查询处理管道的相同阶段的线程。

例如，当从表中读取时，如果可以使用函数来评估表达式，请使用WHERE进行过滤，并且至少使用并行方式对GROUP BY进行预聚合‘max_threads’线程数，然后‘max_threads’被使用。

默认值：物理CPU内核数。

如果一次在服务器上运行的SELECT查询通常少于一个，请将此参数设置为略小于实际处理器内核数的值。

对于由于限制而快速完成的查询，可以设置较低的‘max_threads’。例如，如果必要数量的条目位于每个块中，并且max_threads=8，则会检索8个块，尽管只读取一个块就足够了。

越小 max_threads 值，较少的内存被消耗。

max_insert_threads

要执行的最大线程数 `INSERT SELECT` 查询。

可能的值:

- 0 (or 1) — `INSERT SELECT` 没有并行执行。
- 整数。 大于1。

默认值: 0。

平行 `INSERT SELECT` 只有在 `SELECT` 部分并行执行, 请参阅 `max_threads` 设置。

更高的值将导致更高的内存使用率。

`max_compress_block_size`

在压缩写入表之前, 未压缩数据块的最大大小。默认情况下, 1,048,576 (1MiB)。如果大小减小, 则压缩率显着降低, 压缩和解压缩速度由于高速缓存局部性而略微增加, 并且内存消耗减少。通常没有任何理由更改此设置。

不要将用于压缩的块 (由字节组成的内存块) 与用于查询处理的块 (表中的一组行) 混淆。

`min_compress_block_size`

为 `MergeTree` 表。为了减少处理查询时的延迟, 在写入下一个标记时, 如果块的大小至少为 '`min_compress_block_size`'。默认情况下, 65,536。

块的实际大小, 如果未压缩的数据小于 '`max_compress_block_size`', 是不小于该值且不小于一个标记的数据量。

让我们来看看一个例子。假设 '`index_granularity`' 在表创建期间设置为 8192。

我们正在编写一个 `UInt32` 类型的列 (每个值 4 个字节)。当写入 8192 行时, 总数将是 32KB 的数据。由于 `min_compress_block_size=65,536`, 将为每两个标记形成一个压缩块。

我们正在编写一个字符串类型的 URL 列 (每个值的平均大小 60 字节)。当写入 8192 行时, 平均数据将略少于 500KB。由于这超过 65,536, 将为每个标记形成一个压缩块。在这种情况下, 当从单个标记范围内的磁盘读取数据时, 额外的数据不会被解压缩。

通常没有任何理由更改此设置。

`max_query_size`

查询的最大部分, 可以被带到 RAM 用于使用 SQL 解析器进行解析。

插入查询还包含由单独的流解析器 (消耗 O(1)RAM) 处理的插入数据, 这些数据不包含在此限制中。

默认值: 256KiB。

`interactive_delay`

以微秒为单位的间隔, 用于检查请求执行是否已被取消并发送进度。

默认值: 100,000 (检查取消并每秒发送十次进度)。

`connect_timeout, receive_timeout, send_timeout`

用于与客户端通信的套接字上的超时以秒为单位。

默认值: 10, 300, 300。

`cancel_http_READONLY_queries_on_client_close`

Cancels HTTP read-only queries (e.g. `SELECT`) when a client closes the connection without waiting for the response.

默认值: 0

poll_interval

锁定在指定秒数的等待循环。

默认值：10。

max_distributed_connections

与远程服务器同时连接的最大数量，用于分布式处理对单个分布式表的单个查询。 我们建议设置不小于群集中服务器数量的值。

默认值：1024。

以下参数仅在创建分布式表（以及启动服务器时）时使用，因此没有理由在运行时更改它们。

distributed_connections_pool_size

与远程服务器同时连接的最大数量，用于分布式处理对单个分布式表的所有查询。 我们建议设置不小于群集中服务器数量的值。

默认值：1024。

connect_timeout_with_failover_ms

以毫秒为单位连接到分布式表引擎的远程服务器的超时，如果 ‘shard’ 和 ‘replica’ 部分用于群集定义。
如果不成功，将尝试多次连接到各种副本。

默认值：50。

connections_with_failover_max_tries

分布式表引擎的每个副本的最大连接尝试次数。

默认值：3。

极端

是否计算极值（查询结果列中的最小值和最大值）。 接受0或1。 默认情况下，0（禁用）。

有关详细信息，请参阅部分 “Extreme values”.

use_uncompressed_cache

是否使用未压缩块的缓存。 接受0或1。 默认情况下，0（禁用）。

使用未压缩缓存（仅适用于MergeTree系列中的表）可以在处理大量短查询时显着减少延迟并提高吞吐量。 为频繁发送短请求的用户启用此设置。 还要注意 **uncompressed_cache_size** configuration parameter (only set in the config file) – the size of uncompressed cache blocks. By default, it is 8 GiB. The uncompressed cache is filled in as needed and the least-used data is automatically deleted.

对于至少读取大量数据（一百万行或更多行）的查询，将自动禁用未压缩缓存，以节省真正小型查询的空间。 这意味着你可以保持 ‘use_uncompressed_cache’ 设置始终设置为1。

replace_running_query

当使用HTTP接口时，‘query_id’ 参数可以传递。 这是用作查询标识符的任何字符串。

如果来自同一用户的查询具有相同的 ‘query_id’ 已经存在在这个时候，行为取决于 ‘replace_running_query’ 参数。

0 (default) – Throw an exception (don't allow the query to run if a query with the same ‘query_id’ 已经运行)。

1 – Cancel the old query and start running the new one.

YandexMetrica使用此参数设置为1来实现分段条件的建议。 输入下一个字符后，如果旧的查询还没有完成，应该取消。

stream_flush_interval_ms

适用于在超时的情况下或线程生成流式传输的表 `max_insert_block_size` 行。

默认值为 7500。

值越小，数据被刷新到表中的频率就越高。将该值设置得太低会导致性能较差。

load_balancing

指定用于分布式查询处理的副本选择算法。

ClickHouse 支持以下选择副本的算法：

- 随机（默认情况下）
- 最近的主机名
- 按顺序
- 第一次或随机

随机（默认情况下）

```
load_balancing = random
```

对每个副本计算错误数。查询发送到错误最少的副本，如果存在其中几个错误，则发送给其中任何一个。

缺点：不考虑服务器邻近度；如果副本具有不同的数据，则也会获得不同的数据。

最近的主机名

```
load_balancing = nearest_hostname
```

The number of errors is counted for each replica. Every 5 minutes, the number of errors is integrally divided by 2. Thus, the number of errors is calculated for a recent time with exponential smoothing. If there is one replica with a minimal number of errors (i.e. errors occurred recently on the other replicas), the query is sent to it. If there are multiple replicas with the same minimal number of errors, the query is sent to the replica with a hostname that is most similar to the server's hostname in the config file (for the number of different characters in identical positions, up to the minimum length of both hostnames).

例如，例如 01-01-1 和 example01-01-2.yandex.ru 在一个位置是不同的，而 example01-01-1 和 example01-02-2 在两个地方不同。

这种方法可能看起来很原始，但它不需要有关网络拓扑的外部数据，也不比较 IP 地址，这对于我们的 IPv6 地址来说会很复杂。

因此，如果存在等效副本，则首选按名称最接近的副本。

我们还可以假设，当向同一台服务器发送查询时，在没有失败的情况下，分布式查询也将转到同一台服务器。因此，即使在副本上放置了不同的数据，查询也会返回大多相同的结果。

按顺序

```
load_balancing = in_order
```

具有相同错误数的副本的访问顺序与配置中指定的顺序相同。

当您确切知道哪个副本是可取的时，此方法是适当的。

第一次或随机

```
load_balancing = first_or_random
```

此算法选择集合中的第一个副本，如果第一个副本不可用，则选择随机副本。它在跨复制拓扑设置中有效，但在其他配置中

无用。

该 `first_or_random` 算法解决的问题 `in_order` 算法。与 `in_order`，如果一个副本出现故障，下一个副本将获得双重负载，而其余副本将处理通常的流量。使用时 `first_or_random` 算法中，负载均匀分布在仍然可用的副本之间。

prefer_localhost_replica

在处理分布式查询时，最好使用localhost副本启用/禁用该副本。

可能的值：

- 1 — ClickHouse always sends a query to the localhost replica if it exists.
- 0 — ClickHouse uses the balancing strategy specified by the `load_balancing` 设置。

默认值：1。

警告

如果使用此设置，请禁用此设置 `max_parallel_replicas`.

totals_mode

如何计算总计时有存在，以及当 `max_rows_to_group_by` 和 `group_by_overflow_mode = 'any'` 都在场。
请参阅部分“WITH TOTALS modifier”。

totals_auto_threshold

阈值 `totals_mode = 'auto'`.

请参阅部分“WITH TOTALS modifier”。

max_parallel_replicas

执行查询时每个分片的最大副本数。

为了保持一致性（以获取相同数据拆分的不同部分），此选项仅在设置了采样键时有效。

副本滞后不受控制。

编译

启用查询的编译。默认情况下，0（禁用）。

编译仅用于查询处理管道的一部分：用于聚合的第一阶段（GROUP BY）。

如果编译了管道的这一部分，则由于部署周期较短和内联聚合函数调用，查询可能运行得更快。对于具有多个简单聚合函数的查询，可以看到最大的性能改进（在极少数情况下可快四倍）。通常，性能增益是微不足道的。在极少数情况下，它可能会减慢查询执行速度。

min_count_to_compile

在运行编译之前可能使用已编译代码块的次数。默认情况下，3。

For testing, the value can be set to 0: compilation runs synchronously and the query waits for the end of the compilation process before continuing execution. For all other cases, use values starting with 1. Compilation normally takes about 5-10 seconds.

如果该值为1或更大，则编译在单独的线程中异步进行。结果将在准备就绪后立即使用，包括当前正在运行的查询。

对于查询中使用的聚合函数的每个不同组合以及GROUP BY子句中的键类型，都需要编译代码。

The results of the compilation are saved in the build directory in the form of .so files. There is no restriction on the number of compilation results since they don't use very much space. Old results will be used after server restarts, except in the case of a server upgrade – in this case, the old results are deleted.

output_format_json_quote_64bit_integers

如果该值为 true，则在使用 JSON*Int64 和 UInt64 格式时，整数将显示在引号中（为了与大多数 JavaScript 实现兼容）；否则，整数将不带引号输出。

format_csv_delimiter

将字符解释为 CSV 数据中的分隔符。默认情况下，分隔符为 ,.

input_format_csv_unquoted_null_literal_as_null

对于 CSV 输入格式，启用或禁用未引用的解析 NULL 作为文字（同义词 \N）。

output_format_csv_crlf_end_of_line

在 CSV 中使用 DOS/Windows 样式的行分隔符 (CRLF) 而不是 Unix 样式 (LF)。

output_format_tsv_crlf_end_of_line

在 TSV 中使用 DOC/Windows 样式的行分隔符 (CRLF) 而不是 Unix 样式 (LF)。

insert_quorum

启用仲裁写入。

- 如果 insert_quorum < 2，仲裁写入被禁用。
- 如果 insert_quorum >= 2，仲裁写入已启用。

默认值：0。

仲裁写入

INSERT 只有当 ClickHouse 设法正确地将数据写入成功 insert_quorum 在复制品的 insert_quorum_timeout。如果由于任何原因，成功写入的副本数量没有达到 insert_quorum，写入被认为失败，ClickHouse 将从已经写入数据的所有副本中删除插入的块。

仲裁中的所有副本都是一致的，即它们包含来自所有以前的数据 INSERT 查询。该 INSERT 序列线性化。

当读取从写入的数据 insert_quorum，您可以使用 select_sequential_consistency 选项。

ClickHouse 生成异常

- 如果查询时可用副本的数量小于 insert_quorum。
- 在尝试写入数据时，以前的块尚未被插入 insert_quorum 的复制品。如果用户尝试执行 INSERT 前一个与 insert_quorum 完成。

另请参阅：

- insert_quorum_timeout
- select_sequential_consistency

insert_quorum_timeout

写入仲裁超时以秒为单位。如果超时已经过去，并且还没有发生写入，ClickHouse 将生成异常，客户端必须重复查询以将相同的块写入相同的副本或任何其他副本。

默认值：60 秒。

另请参阅：

- insert_quorum
- select_sequential_consistency

select_sequential_consistency

启用或禁用顺序一致性 `SELECT` 查询:

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 0。

用途

当启用顺序一致性时，ClickHouse允许客户端执行 `SELECT` 仅查询那些包含来自所有先前数据的副本 `INSERT` 查询执行 `insert_quorum`. 如果客户端引用了部分副本，ClickHouse将生成异常。 `SELECT`查询将不包括尚未写入副本仲裁的数据。

另请参阅:

- [insert_quorum](#)
- [insert_quorum_timeout](#)

insert_deduplicate

启用或禁用块重复数据删除 `INSERT` (对于复制的*表)。

可能的值:

- 0 — Disabled.
- 1 — Enabled.

默认值 : 1。

默认情况下，块插入到复制的表 `INSERT` 语句重复数据删除 (见 [数据复制](#)).

deduplicate_blocks_in_dependent_materialized_views

启用或禁用从已复制*表接收数据的实例化视图的重复数据删除检查。

可能的值:

- | |
|-------------------------------|
| 0 — Disabled.
1 — Enabled. |
|-------------------------------|

默认值 : 0。

用途

默认情况下，重复数据删除不对实例化视图执行，而是在源表的上游执行。

如果由于源表中的重复数据删除而跳过了插入的块，则不会插入附加的实例化视图。这种行为的存在是为了允许将高度聚合的数据插入到实例化视图中，对于在实例化视图聚合之后插入的块相同，但是从源表中的不同插入派生的情况。

与此同时，这种行为“breaks”`INSERT` 署等性 如果一个 `INSERT` 进入主表是成功的，`INSERT into a materialized view failed` (e.g. because of communication failure with Zookeeper) a client will get an error and can retry the operation. However, the materialized view won't receive the second insert because it will be discarded by deduplication in the main (source) table. The setting `deduplicate_blocks_in_dependent_materialized_views` 允许改变这种行为。重试时，实例化视图将收到重复插入，并自行执行重复数据删除检查，忽略源表的检查结果，并将插入由于第一次失败而丢失的行。

max_network_bytes

限制在执行查询时通过网络接收或传输的数据量 (以字节为单位)。此设置适用于每个单独的查询。

可能的值:

- 整数。
- 0 — Data volume control is disabled.

默认值：0。

max_network_bandwidth

限制通过网络进行数据交换的速度，以每秒字节为单位。此设置适用于每个查询。

可能的值：

- 整数。
- 0 — Bandwidth control is disabled.

默认值：0。

max_network_bandwidth_for_user

限制通过网络进行数据交换的速度，以每秒字节为单位。此设置适用于单个用户执行的所有并发运行的查询。

可能的值：

- 整数。
- 0 — Control of the data speed is disabled.

默认值：0。

max_network_bandwidth_for_all_users

限制通过网络交换数据的速度，以每秒字节为单位。此设置适用于服务器上同时运行的所有查询。

可能的值：

- 整数。
- 0 — Control of the data speed is disabled.

默认值：0。

count_distinct_implementation

指定其中的 `uniq*` 函数应用于执行 `COUNT(DISTINCT ...)` 建筑。

可能的值：

- `uniq`
- `uniqCombined`
- `uniqCombined64`
- `uniqHLL12`
- `uniqExact`

默认值：`uniqExact`。

skip_unavailable_shards

启用或禁用静默跳过不可用分片。

如果分片的所有副本都不可用，则视为不可用。副本在以下情况下不可用：

- ClickHouse出于任何原因无法连接到副本。

连接到副本时，ClickHouse会执行多次尝试。如果所有这些尝试都失败，则认为副本不可用。

- 副本无法通过DNS解析。

如果无法通过DNS解析副本的主机名，则可能指示以下情况：

- 副本的主机没有DNS记录。它可以发生在具有动态DNS的系统中，例如，Kubernetes，其中节点在停机期间可能无法解决问题，这不是错误。
- 配置错误。ClickHouse配置文件包含错误的主机名。

可能的值：

- 1 — skipping enabled.

如果分片不可用，ClickHouse将基于部分数据返回结果，并且不报告节点可用性问题。

- 0 — skipping disabled.

如果分片不可用，ClickHouse将引发异常。

默认值：0。

optimize_skip_unused_shards

对于在PREWHERE/WHERE中具有分片键条件的SELECT查询，启用或禁用跳过未使用的分片（假定数据是通过分片键分发的，否则不执行任何操作）。

默认值：0

force_optimize_skip_unused_shards

在以下情况下启用或禁用查询执行 `optimize_skip_unused_shards` 无法启用和跳过未使用的分片。如果跳过是不可能的，并且设置为启用异常将被抛出。

可能的值：

- 0-禁用（不抛出）
- 1-仅当表具有分片键时禁用查询执行
- 2-无论为表定义了分片键，都禁用查询执行

默认值：0

force_optimize_skip_unused_shards_no_nested

重置 `optimize_skip_unused_shards` 对于嵌套 Distributed 表

可能的值：

- 1 — Enabled.
- 0 — Disabled.

默认值：0。

optimize_throw_if_noop

启用或禁用抛出异常，如果 `OPTIMIZE` 查询未执行合并。

默认情况下，`OPTIMIZE` 即使它没有做任何事情，也会成功返回。此设置允许您区分这些情况并在异常消息中获取原因。

可能的值：

- 1 — Throwing an exception is enabled.
- 0 — Throwing an exception is disabled.

默认值：0。

`distributed_replica_error_half_life`

- 类型：秒
- 默认值：60秒

控制清零分布式表中的错误的速度。如果某个副本在一段时间内不可用，累计出现5个错误，并且`distributed_replica_error_half_life`设置为1秒，则该副本在上一个错误发生3秒后视为正常。

另请参阅：

- 表引擎分布式
- `distributed_replica_error_cap`

`distributed_replica_error_cap`

- 类型：无符号int
- 默认值：1000

每个副本的错误计数上限为此值，从而防止单个副本累积太多错误。

另请参阅：

- 表引擎分布式
- `distributed_replica_error_half_life`

`distributed_directory_monitor_sleep_time_ms`

对于基本间隔 分布 表引擎发送数据。在发生错误时，实际间隔呈指数级增长。

可能的值：

- 毫秒的正整数。

默认值：100毫秒。

`distributed_directory_monitor_max_sleep_time_ms`

的最大间隔 分布 表引擎发送数据。限制在设置的区间的指数增长 `distributed_directory_monitor_sleep_time_ms` 设置。

可能的值：

- 毫秒的正整数。

默认值：30000毫秒（30秒）。

`distributed_directory_monitor_batch_inserts`

启用/禁用批量发送插入的数据。

当批量发送被启用时， 分布 表引擎尝试在一个操作中发送插入数据的多个文件，而不是单独发送它们。批量发送通过更好地利用服务器和网络资源来提高集群性能。

可能的值：

- 1 — Enabled.
- 0 — Disabled.

默认值：0。

`os_thread_priority`

设置优先级（不错）对于执行查询的线程。当选择要在每个可用CPU内核上运行的下一个线程时，操作系统调度程序会考虑此优先级。

警告

要使用此设置，您需要设置 `CAP_SYS_NICE` 能力。该 `clickhouse-server` 软件包在安装过程中设置它。某些虚拟环境不允许您设置 `CAP_SYS_NICE` 能力。在这种情况下，`clickhouse-server` 在开始时显示关于它的消息。

可能的值:

- 您可以在范围内设置值 [-20, 19].

值越低意味着优先级越高。低螺纹 `nice` 与具有高值的线程相比，优先级值的执行频率更高。高值对于长时间运行的非交互式查询更为可取，因为这使得它们可以在到达时快速放弃资源，转而使用短交互式查询。

默认值：0。

query_profiler_real_time_period_ns

设置周期的实时时钟定时器 [查询探查器](#). 真正的时钟计时器计数挂钟时间。

可能的值:

- 正整数，以纳秒为单位。

推荐值:

```
- 10000000 (100 times a second) nanoseconds and less for single queries.  
- 1000000000 (once a second) for cluster-wide profiling.
```

- 0 用于关闭计时器。

类型: [UInt64](#).

默认值: 1000000000 纳秒 (每秒一次)。

另请参阅:

- 系统表 [trace_log](#)

query_profiler_cpu_time_period_ns

设置周期的CPU时钟定时器 [查询探查器](#). 此计时器仅计算CPU时间。

可能的值:

- 纳秒的正整数。

推荐值:

```
- 10000000 (100 times a second) nanoseconds and more for single queries.  
- 1000000000 (once a second) for cluster-wide profiling.
```

- 0 用于关闭计时器。

类型: [UInt64](#).

默认值: 1000000000 纳秒。

另请参阅:

- 系统表 [trace_log](#)

allow_introspection_functions

启用禁用 反省函数 用于查询分析。

可能的值:

- 1 — Introspection functions enabled.
- 0 — Introspection functions disabled.

默认值 : 0。

另请参阅

- 采样查询探查器
- 系统表 `trace_log`

input_format_parallel_parsing

- 类型 : 布尔
- 默认值 : True

启用数据格式的保序并行分析。仅支持TSV，TKSV，CSV和JSONEachRow格式。

min_chunk_bytes_for_parallel_parsing

- 类型 : 无符号int
- 默认值 : 1MiB

以字节为单位的最小块大小，每个线程将并行解析。

output_format_avro_codec

设置用于输出Avro文件的压缩编解码器。

类型 : 字符串

可能的值:

- null — No compression
- deflate — Compress with Deflate (zlib)
- snappy — Compress with **活泼的**

默认值: `snappy` (如果可用) 或 `deflate`.

output_format_avro_sync_interval

设置输出Avro文件的同步标记之间的最小数据大小 (以字节为单位)。

类型 : 无符号int

可能的值 : 32 (32字节) -1073741824 (1GiB)

默认值 : 32768 (32KiB)

format_avro_schema_registry_url

设置要与之一起使用的汇合架构注册表URL **AvroConfluent** 格式

类型 : 网址

默认值 : 空

background_pool_size

设置在表引擎中执行后台操作的线程数（例如，合并 MergeTree 引擎 表）。此设置在 ClickHouse 服务器启动时应用，不能在用户会话中更改。通过调整此设置，您可以管理 CPU 和磁盘负载。较小的池大小使用较少的 CPU 和磁盘资源，但后台进程推进速度较慢，最终可能会影响查询性能。

可能的值：

- 任何正整数。

默认值：16。

原始文章

ツ环板-ヨツ嘉ツツ偲

该 clickhouse-local 程序使您能够对本地文件执行快速处理，而无需部署和配置 ClickHouse 服务器。

接受表示表的数据并使用以下方式查询它们 **ツ环板ECTヨツ嘉ツツ偲**。

clickhouse-local 使用与 ClickHouse server 相同的核心，因此它支持大多数功能以及相同的格式和表引擎。

默认情况下 clickhouse-local 不能访问同一主机上的数据，但它支持使用以下方式加载服务器配置 --config-file 争论。

警告

不建议将生产服务器配置加载到 clickhouse-local 因为数据可以在人为错误的情况下被损坏。

用途

基本用法：

```
clickhouse-local --structure "table_structure" --input-format "format_of_incoming_data" -q "query"
```

参数：

- S, --structure — table structure for input data.
- if, --input-format — input format, TSV 默认情况下。
- f, --file — path to data, stdin 默认情况下。
- q, --query — queries to execute with ; 如 delimiter。
- N, --table — table name where to put output data, table 默认情况下。
- of, --format, --output-format — output format, TSV 默认情况下。
- stacktrace — whether to dump debug output in case of exception.
- verbose — more details on query execution.
- s — disables stderr 记录。
- config-file — path to configuration file in same format as for ClickHouse server, by default the configuration empty.
- help — arguments references for clickhouse-local.

还有每个 ClickHouse 配置变量的参数，这些变量更常用，而不是 --config-file。

例

```
echo -e "1,2\n3,4" | clickhouse-local -S "a Int64, b Int64" -if "CSV" -q "SELECT * FROM table"
Read 2 rows, 32.00 B in 0.000 sec., 5182 rows/sec., 80.97 KiB/sec.
1 2
3 4
```

前面的例子是一样的：

```
$ echo -e "1,2\n3,4" | clickhouse-local -q "CREATE TABLE table (a Int64, b Int64) ENGINE = File(CSV, stdin); SELECT a, b FROM table; DROP TABLE table"
Read 2 rows, 32.00 B in 0.000 sec., 4987 rows/sec., 77.93 KiB/sec.
1 2
3 4
```

现在让我们为每个Unix用户输出内存用户：

```
$ ps aux | tail -n +2 | awk '{ printf("%s\t%s\n", $1, $4) }' | clickhouse-local -S "user String, mem Float64" -q "SELECT user, round(sum(mem), 2) as memTotal FROM table GROUP BY user ORDER BY memTotal DESC FORMAT Pretty"
Read 186 rows, 4.15 KiB in 0.035 sec., 5302 rows/sec., 118.34 KiB/sec.
```

user	memTotal
bayonet	113.5
root	8.8
...	

ツ暗エツ汎环催ツ団

连接到ClickHouse服务器并重复发送指定的查询。

语法：

```
$ echo "single query" | clickhouse-benchmark [keys]
```

或

```
$ clickhouse-benchmark [keys] <<< "single query"
```

如果要发送一组查询，请创建一个文本文件，并将每个查询放在此文件中的单个字符串上。例如：

```
SELECT * FROM system.numbers LIMIT 10000000
SELECT 1
```

然后将此文件传递给标准输入 `clickhouse-benchmark`.

```
clickhouse-benchmark [keys] < queries_file
```

键

- `-c N, --concurrency=N` — Number of queries that `clickhouse-benchmark` 同时发送。默认值：1。
- `-d N, --delay=N` — Interval in seconds between intermediate reports (set 0 to disable reports). Default value: 1.
- `-h WORD, --host=WORD` — Server host. Default value: `localhost`. 为 比较模式 您可以使用多个 `-h` 钥匙
- `-p N, --port=N` — Server port. Default value: 9000. For the 比较模式 您可以使用多个 `-p` 钥匙
- `--iterations=N` — Total number of queries. Default value: 0

- `-I N, --iterations=N` — Total number of queries. Default value: 0.
- `-r, --randomize` — Random order of queries execution if there is more than one input query.
- `-s, --secure` — Using TLS connection.
- `-t N, --timelimit=N` — Time limit in seconds. `clickhouse-benchmark` 达到指定的时间限制时停止发送查询。默认值: 0 (禁用时间限制)。
- `--confidence=N` — Level of confidence for T-test. Possible values: 0 (80%), 1 (90%), 2 (95%), 3 (98%), 4 (99%), 5 (99.5%). Default value: 5. In the 比较模式 `clickhouse-benchmark` 执行 独立双样本学生的t测试 测试以确定两个分布是否与所选置信水平没有不同。
- `--cumulative` — Printing cumulative data instead of data per interval.
- `--database=DATABASE_NAME` — ClickHouse database name. Default value: `default`.
- `--json=FILEPATH` — JSON output. When the key is set, `clickhouse-benchmark` 将报告输出到指定的JSON文件。
- `--user=USERNAME` — ClickHouse user name. Default value: `default`.
- `--password=PSWD` — ClickHouse user password. Default value: empty string.
- `--stacktrace` — Stack traces output. When the key is set, `clickhouse-benchmark` 输出异常的堆栈跟踪。
- `--stage=WORD` — Query processing stage at server. ClickHouse stops query processing and returns answer to `clickhouse-benchmark` 在指定的阶段。可能的值: `complete`, `fetch_columns`, `with_mergeable_state`. 默认值: `complete`.
- `--help` — Shows the help message.

如果你想申请一些 设置 对于查询，请将它们作为键传递 `--<session setting name>= SETTING_VALUE`. 例如, `--max_memory_usage=1048576`.

输出

默认情况下, `clickhouse-benchmark` 每个报表 `--delay` 间隔。

报告示例:

```
Queries executed: 10.
```

```
localhost:9000, queries 10, QPS: 6.772, RPS: 67904487.440, MiB/s: 518.070, result RPS: 67721584.984, result MiB/s: 516.675.
```

```
0.000% 0.145 sec.
10.000% 0.146 sec.
20.000% 0.146 sec.
30.000% 0.146 sec.
40.000% 0.147 sec.
50.000% 0.148 sec.
60.000% 0.148 sec.
70.000% 0.148 sec.
80.000% 0.149 sec.
90.000% 0.150 sec.
95.000% 0.150 sec.
99.000% 0.150 sec.
99.900% 0.150 sec.
99.990% 0.150 sec.
```

在报告中，您可以找到:

- 在查询的数量 `Queries executed:` 场。
- 状态字符串包含 (按顺序):
 - ClickHouse服务器的端点。
 - 已处理的查询数。
 - QPS : QPS : 在指定的时间段内每秒执行多少个查询服务器 `--delay` 争论。
 - RPS : 在指定的时间段内，服务器每秒读取多少行 `--delay` 争论。
 - MiB/s : 在指定的时间段内每秒读取多少mebibytes服务器 `--delay` 争论。

- 结果RPS：在指定的时间段内，服务器每秒放置到查询结果的行数 `--delay` 争论。
- 结果MiB/s. 在指定的时间段内，服务器每秒将多少mebibytes放置到查询结果中 `--delay` 争论。
- 查询执行时间的百分位数。

比较模式

`clickhouse-benchmark` 可以比较两个正在运行的ClickHouse服务器的性能。

要使用比较模式，请通过以下两对指定两个服务器的端点 `--host`, `--port` 钥匙键在参数列表中的位置匹配在一起，第一 `--host` 与第一匹配 `--port` 等等。`clickhouse-benchmark` 建立到两个服务器的连接，然后发送查询。每个查询寻址到随机选择的服务器。每个服务器的结果分别显示。

示例

```
$ echo "SELECT * FROM system.numbers LIMIT 10000000 OFFSET 10000000" | clickhouse-benchmark -i 10
```

```
Loaded 1 queries.
```

```
Queries executed: 6.
```

```
localhost:9000, queries 6, QPS: 6.153, RPS: 123398340.957, MiB/s: 941.455, result RPS: 61532982.200, result MiB/s: 469.459.
```

```
0.000% 0.159 sec.  
10.000% 0.159 sec.  
20.000% 0.159 sec.  
30.000% 0.160 sec.  
40.000% 0.160 sec.  
50.000% 0.162 sec.  
60.000% 0.164 sec.  
70.000% 0.165 sec.  
80.000% 0.166 sec.  
90.000% 0.166 sec.  
95.000% 0.167 sec.  
99.000% 0.167 sec.  
99.900% 0.167 sec.  
99.990% 0.167 sec.
```

```
Queries executed: 10.
```

```
localhost:9000, queries 10, QPS: 6.082, RPS: 121959604.568, MiB/s: 930.478, result RPS: 60815551.642, result MiB/s: 463.986.
```

```
0.000% 0.159 sec.  
10.000% 0.159 sec.  
20.000% 0.160 sec.  
30.000% 0.163 sec.  
40.000% 0.164 sec.  
50.000% 0.165 sec.  
60.000% 0.166 sec.  
70.000% 0.166 sec.  
80.000% 0.167 sec.  
90.000% 0.167 sec.  
95.000% 0.170 sec.  
99.000% 0.172 sec.  
99.900% 0.172 sec.  
99.990% 0.172 sec.
```

clickhouse-copier

将数据从一个群集中的表复制到另一个（或相同）群集中的表。

您可以运行多个 `clickhouse-copier` 不同服务器上的实例执行相同的作业。ZooKeeper用于同步进程。

开始后，`clickhouse-copier`:

- 连接到ZooKeeper并且接收：
 - 复制作业。
 - 复制作业的状态。
- 它执行的工作。

每个正在运行的进程都会选择源集群的“最接近”分片，然后将数据复制到目标集群，并在必要时重新分片数据。

`clickhouse-copier` 跟踪ZooKeeper中的更改，并实时应用它们。

为了减少网络流量，我们建议运行 `clickhouse-copier` 在源数据所在的同一服务器上。

运行Clickhouse-copier

该实用程序应手动运行:

```
clickhouse-copier copier --daemon --config zookeeper.xml --task-path /task/path --base-dir /path/to/dir
```

参数:

- `daemon` — 在守护进程模式下启动 `clickhouse-copier`。
- `config` — `zookeeper.xml`文件的路径，其中包含用于连接ZooKeeper的参数。
- `task-path` — ZooKeeper节点的路径。该节点用于同步 `clickhouse-copier` 进程和存储任务。任务存储在 `$task-path/description` 中。
- `task-file` — 可选的非必须参数，指定一个包含任务配置的参数文件，用于初始上传到ZooKeeper。
- `task-upload-force` — 即使节点已经存在，也强制上载 `task-file`。
- `base-dir` — 日志和辅助文件的路径。启动时，`clickhouse-copier`在 `$base-dir` 中创建 `clickhouse-copier_YYYYMMHHSS_<PID>` 子目录。如果省略此参数，则会在启动 `clickhouse-copier` 的目录中创建目录。

Zookeeper.xml格式

```
<yandex>
  <logger>
    <level>trace</level>
    <size>100M</size>
    <count>3</count>
  </logger>

  <zookeeper>
    <node index="1">
      <host>127.0.0.1</host>
      <port>2181</port>
    </node>
  </zookeeper>
</yandex>
```

复制任务的配置

```

<yandex>
  <!-- Configuration of clusters as in an ordinary server config -->
  <remote_servers>
    <source_cluster>
      <shard>
        <internal_replication>false</internal_replication>
        <replica>
          <host>127.0.0.1</host>
          <port>9000</port>
        </replica>
      </shard>
      ...
    </source_cluster>

    <destination_cluster>
    ...
  </destination_cluster>
</remote_servers>

  <!-- How many simultaneously active workers are possible. If you run more workers superfluous workers will sleep. -->
  >
  <max_workers>2</max_workers>

  <!-- Setting used to fetch (pull) data from source cluster tables -->
  <settings_pull>
    <readonly>1</readonly>
  </settings_pull>

  <!-- Setting used to insert (push) data to destination cluster tables -->
  <settings_push>
    <readonly>0</readonly>
  </settings_push>

  <!-- Common setting for fetch (pull) and insert (push) operations. Also, copier process context uses it.
       They are overlaid by <settings_pull/> and <settings_push/> respectively. -->
  <settings>
    <connect_timeout>3</connect_timeout>
    <!-- Sync insert is set forcibly, leave it here just in case. -->
    <insert_distributed_sync>1</insert_distributed_sync>
  </settings>

  <!-- Copying tasks description.
       You could specify several table task in the same task description (in the same ZooKeeper node), they will be
       performed
       sequentially.
     -->
  <tables>
    <!-- A table task, copies one table. -->
    <table_hits>
      <!-- Source cluster name (from <remote_servers/> section) and tables in it that should be copied -->
      <cluster_pull>source_cluster</cluster_pull>
      <database_pull>test</database_pull>
      <table_pull>hits</table_pull>

      <!-- Destination cluster name and tables in which the data should be inserted -->
      <cluster_push>destination_cluster</cluster_push>
      <database_push>test</database_push>
      <table_push>hits2</table_push>

      <!-- Engine of destination tables.
           If destination tables have not be created, workers create them using columns definition from source tables
           and engine
         -->
    <definition_from_here>
  </tables>

```

definition from here.

NOTE: If the first worker starts insert data and detects that destination partition is not empty then the partition will

be dropped and refilled, take it into account if you already have some data in destination tables. You could directly

specify partitions that should be copied in <enabled_partitions/>, they should be in quoted format like partition column of system.parts table.

-->

<engine>

```
ENGINE=ReplicatedMergeTree('/clickhouse/tables/{cluster}/{shard}/hits2', '{replica}')
PARTITION BY toMonday(date)
ORDER BY (CounterID, EventDate)
</engine>
```

<!-- Sharding key used to insert data to destination cluster -->

```
<sharding_key>jumpConsistentHash(intHash64(UserID), 2)</sharding_key>
```

<!-- Optional expression that filter data while pull them from source servers -->

```
<where_condition>CounterID != 0</where_condition>
```

<!-- This section specifies partitions that should be copied, other partition will be ignored.

Partition names should have the same format as

partition column of system.parts table (i.e. a quoted text).

Since partition key of source and destination cluster could be different,

these partition names specify destination partitions.

NOTE: In spite of this section is optional (if it is not specified, all partitions will be copied), it is strictly recommended to specify them explicitly.

If you already have some ready partitions on destination cluster they will be removed at the start of the copying since they will be interpreted as unfinished data from the previous copying!!!

-->

<enabled_partitions>

```
<partition>'2018-02-26'</partition>
```

```
<partition>'2018-03-05'</partition>
```

...

</enabled_partitions>

</table_hits>

<!-- Next table to copy. It is not copied until previous table is copying. -->

</table_visits>

...

</table_visits>

...

</tables>

</yandex>

clickhouse-copier 跟踪更改 /task/path/description 并在飞行中应用它们。例如，如果你改变的值 max_workers ，运行任务的进程数也会发生变化。

ツ環板 Utility ヨツ嘉ツ

- ツ環板-ヨツ嘉ツ専 — Allows running SQL queries on data without stopping the ClickHouse server, similar to how awk 做到这一点。
- ツ環板-ヨツ嘉ツ専 — Copies (and reshards) data from one cluster to another cluster.
- ツ暗エツ汎環催ツ団 — Loads server with the custom queries and settings.

这个内容有帮助??

RATING_STARS

评分: RATING_VALUE - RATING_COUNT 所得票数

©2016-2020 Yandex LLC

建于 8eac46c79f